

Around Context-Free Grammars - a Normal Form, a Representation Theorem, and a Regular Approximation

Liliana Cojocaru

School of Information Sciences, Computer Science

University of Tampere

Liliana.Cojocaru@uta.fi

Abstract

We introduce a normal form for *context-free grammars*, called *Dyck normal form*. This is a syntactical restriction of the *Chomsky normal form*, in which the two nonterminals occurring on the right-hand side of a rule are paired nonterminals. This pairwise property allows to define a homomorphism from Dyck words to words generated by a grammar in Dyck normal form. We prove that for each context-free language L , there exist an integer K and a homomorphism φ such that $L = \varphi(D'_K)$, where $D'_K \subseteq D_K$, and D_K is the one-sided Dyck language over K letters. Through a *transition-like diagram* for a context-free grammar in Dyck normal form, we effectively build a regular language R such that $D'_K = R \cap D_K$, which leads to the *Chomsky-Schützenberger theorem*. Using graphical approaches we refine R such that the Chomsky-Schützenberger theorem still holds. Based on this readjustment we sketch a *transition diagram* for a *regular grammar* that generates a *regular superset approximation* for the initial context-free language.

Keywords: linear languages, context-free languages, Dyck languages, Chomsky normal form, Dyck normal form, Chomsky-Schützenberger theorem, regular approximation

Introduction

A *normal form* for context-free grammars consists of restrictions imposed on the structure of grammar's productions. These restrictions concern the number of terminals and nonterminals allowed on the right-hand sides of the rules, or on the manner in which terminals and nonterminals are arranged into the rules. Normal forms turned out to be useful tools in studying syntactical properties of context-free grammars, in parsing theory, structural and descriptive complexity, inference and learning theory. Various normal forms for context-free grammars have been studied so far, but the most important remain the Chomsky normal form [17], Greibach normal form [12], and operator normal form [17]. For definitions, results, and surveys on normal forms the reader is referred to [5], [17], and [20]. A normal form is correct if it preserves the language generated by the original grammar. This condition is called *the weak equivalence*, i.e., a normal form preserves the language but may lose important syntactical or semantical properties of the original grammar. The more syntactical, semantical, or ambiguity properties a normal form preserves, the stronger it is. It is well known that the Chomsky normal form is a *strong* normal form.

This paper is partly devoted to a new normal form for context-free grammars, called *Dyck normal form*. The Dyck normal form is a syntactical restriction of the Chomsky normal form, in which the two nonterminals occurring on the right-hand side of a rule are paired nonterminals, in the sense that each left (right) nonterminal of a pair has a unique right (left) pairwise. This pairwise property imposed on the structure of the right-hand side of each rule induces a nested structure on the derivation tree of each word generated by a grammar in Dyck normal form. More precisely, each derivation tree of a word generated by a grammar in Dyck normal form, that is read in the depth-first search order is a Dyck word, hence the name of the normal form. Furthermore, there exists always a homomorphism between the derivation tree of a word generated by a grammar in Chomsky normal form and its equivalent in Dyck normal form. In other words the Chomsky and Dyck normal forms are *strongly equivalent*. This property, along with several other terminal rewriting conditions imposed to a grammar in Dyck normal form, enable us to define a homomorphism from Dyck words to words generated by a grammar in Dyck normal form. We have been inspired to develop this normal form by the general theory of Dyck words and Dyck languages, that turned out to play a crucial role in the description and characterization of context-free languages [9], [10], and [19]. The definition and several properties of grammars in Dyck normal form are presented in Section 1.

For each context-free grammar G in Dyck normal form we define, in Section 2, the *trace language* associated with derivations in G , which is the set of all derivation trees of G read in the depth-first search order, starting from the grammar axiom. By exploiting the Dyck normal form, and several characterizations of Dyck languages presented in [19], we give a new characterization of context-free languages in terms of Dyck languages. We prove (also in Section 2) that for each context-free language L , generated by a grammar G in Dyck normal form, there exist an integer K and a homomorphism φ such that $L = \varphi(D'_K)$, where D'_K (a subset of the Dyck language over K letters) equals, with very little exceptions, the trace language associated with G .

In Section 3 we show that the representation theorem in Section 2 emerges, through a *transition-like diagram* for context-free grammars in Dyck normal form, to the Chomsky-Schützenberger theorem. By improving this transition diagram, in Section 4 we refine the regular language provided by the Chomsky-Schützenberger theorem, while in Section 5 we show that the refined graphical representation of derivations in a context-free grammar in Dyck normal form, used in the previous sections, provides a framework for a regular grammar that generates a *regular superset approximation* for the initial context-free language.

The method used throughout this paper is graph-constructive, in the sense that it supplies a graphical interpretation of the Chomsky-Schützenberger theorem, and consequently it shows how to graphically build a regular language (as minimal as possible) that satisfies this theorem. Even if we reach the same famous Chomsky-Schützenberger theorem, the method used to approach it is different from the other methods known in the literature. In brief, the method in [17] is based on pushdown approaches, while that in [11] uses some kind of imaginary brackets that simulate the work of a pushdown store, when deriving a context-free language. The method presented in [1] uses equations on languages and algebraical approaches to derive several types of Dyck language generators for context-free languages. In all these works, the Dyck language is somehow hidden behind the deriva-

tive structure of the context-free language (supplementary brackets are needed to derive a Dyck language generator for a context-free language). The Dyck language provided in this paper is merely found through a *pairwise-renaming procedure* of the nonterminals in the original context-free grammar. Hence, it lies inside the context-free grammar it describes. Each method used in the literature to prove the Chomsky-Schützenberger theorem provides its own regular language. Our aim is to find a thinner regular language that satisfies the Chomsky-Schützenberger theorem (with respect to the method hereby used) and approaching this language to build a regular superset approximation for context-free languages (likely to be as thinner as possible).

Note that the concept of a thinner (minimal) regular language, for the Chomsky-Schützenberger theorem and for the regular superset approximation is *relative*, in the sense that it depends on the structure of the grammar in Dyck normal form used to generate the original context-free language. In [2], [14], [15], and [16] it is proved that there is no algorithm that builds, for an arbitrary context-free language L , the minimal context-free grammar that generates L , where the minimality of a context-free grammar is considered, in principal, with respect to descriptonal measures such as number of nonterminals, rules, and loops (i.e., grammatical levels [14], encountered during derivations in a context-free grammar). Consequently, there is no algorithm to build a minimal regular superset approximation for an arbitrary context-free language. Attempts to find optimal regular superset (subsets) approximations for context-free languages can be found in [4], [6], [21], and [23]. In Sections 3, 4, and 5 we also illustrate, through several examples, the manner in which the regular languages provided by the Chomsky-Schützenberger theorem and by the regular approximation can be built, with regards to the method proposed in this paper.

1 Dyck Normal Form

We assume the reader to be familiar with the basic notions of formal language theory [17]. For an alphabet X , X^* denotes the free monoid generated by X . By $|x|_a$ we denote the number of occurrences of the letter a in the string $x \in X^*$, while $|x|$ is the length of $x \in V^*$. We denote by λ the empty string. If X is a finite set, then $|X|$ is the cardinality of X .

Definition 1.1. A context-free grammar¹ $G = (N, T, P, S)$ is said to be in *Dyck normal form* if it satisfies the following conditions:

1. G is in Chomsky normal form,
2. if $A \rightarrow a \in P$, $A \in N$, $A \neq S$, $a \in T$, then no other rule in P rewrites A ,
3. for each $A \in N$ such that $X \rightarrow AB \in P$ ($X \rightarrow BA \in P$) there is no other rule in P of the form $X' \rightarrow B'A$ ($X' \rightarrow AB'$),
4. for each rules $X \rightarrow AB$, $X' \rightarrow A'B$ ($X \rightarrow AB$, $X' \rightarrow AB'$), we have $A = A'$ ($B = B'$).

¹A context-free grammar is denoted by $G = (N, T, P, S)$, where N and T are finite sets of *variables* and *terminals*, respectively, $N \cap T = \emptyset$, $S \in N - T$ is the grammar *axiom*, and $P \subseteq N \times (N \cup T)^*$ is the finite set of *productions*.

Note that the reasons for which we introduce the restrictions at items 2 – 4, are the following. The condition at item 2 allows to make a partition between those nonterminals rewritten by nonterminals, and those nonterminals rewritten by terminals (with the exception of the axiom). This enables, in Section 2, to define a homomorphism from Dyck words to words generated by a grammar in Dyck normal form. Conditions at items 3 and 4 allow to split the set of nonterminals into pairwise nonterminals, and thus to introduce bracketed pairs. The next theorem proves that the Dyck normal form is correct.

Theorem 1.2. *For each context-free grammar $G = (N, T, P, S)$ there exists a grammar $G' = (N', T, P', S)$ such that $L(G) = L(G')$ where G' is in Dyck normal form.*

Proof. Suppose that G is a context-free grammar in Chomsky normal form. Otherwise, using the algorithm described in [20] we can convert G into Chomsky normal form. To convert G from Chomsky normal form into Dyck normal form we proceed as follows.

Step 1 We check whether P contains two (or more) rules of the form $A \rightarrow a$, $A \rightarrow b$, $a \neq b$. If it does, then for each rule $A \rightarrow b$, $a \neq b$, a new variable A_b is introduced. We add the new rule $A_b \rightarrow b$, and remove $A \rightarrow b$. For each rule $X \rightarrow AB$ ($X \rightarrow BA$) we add the new rule $X \rightarrow A_b B$ ($X \rightarrow B A_b$), while for a rule of the form $X \rightarrow AA$ we add three new rules $X \rightarrow A_b A$, $X \rightarrow A A_b$, $X \rightarrow A_b A_b$, without removing the initial rules. We call this procedure an A_b -terminal substitution of A . For each rule $A \rightarrow a$, $a \in T$, we check whether a rule of the form $A \rightarrow B_1 B_2$, $B_1, B_2 \in N$, exists in P . If it does, then a new nonterminal A_a is introduced and an A_a -terminal substitution of A for the rule $A \rightarrow a$ is performed.

Step 2 Suppose there exist two (or more) rules of the form $X \rightarrow AB$ and $X' \rightarrow B'A$. If we have agreed on preserving only the left occurrences of A on the right-hand sides, then according to condition 3 of Definition 1.1, we have to remove all right occurrences of A . To do so we introduce a new nonterminal ${}_Z A$ and all right occurrences of A , preceded at the left side by Z , in the right-hand side of a rule, are substituted by ${}_Z A$. For each rule that rewrites A , $A \rightarrow Y$, $Y \in N^2 \cup T$, we add a new rule of the form ${}_Z A \rightarrow Y$, preserving the rule $A \rightarrow Y$. We call this procedure an ${}_Z A$ -nonterminal substitution of A . According to this procedure, for the rule $X' \rightarrow B'A$, we introduce a new nonterminal ${}_{B'} A$, we add the rule $X' \rightarrow B' {}_{B'} A$, and remove the rule $X' \rightarrow B'A$. For each rule that rewrites A , of the form² $A \rightarrow Y$, $Y \in N^2 \cup T$, we add a new rule of the form ${}_{B'} A \rightarrow Y$, preserving the rule $A \rightarrow Y$.

Step 3 Finally, for each two rules $X \rightarrow AB$, $X' \rightarrow A'B$ ($X \rightarrow BA$, $X' \rightarrow BA'$) with $A \neq A'$, a new nonterminal ${}_{A'} B$ ($B_{A'}$) is introduced to replace B from the second rule, and we perform an ${}_{A'} B$ ($B_{A'}$)-nonterminal substitution of B , i.e., we add $X' \rightarrow A' {}_{A'} B$, and remove $X' \rightarrow A'B$. For each rule that rewrites B , of the form $B \rightarrow Y$, $Y \in N^2 \cup T$, we add a new rule ${}_{A'} B \rightarrow Y$, preserving $B \rightarrow Y$. In the case that A' occurs on the right-hand side of another rule, such that A' matches at the right side with another nonterminal different of ${}_{A'} B$, then the procedure described above is repeated for A' , too.

Note that, if one of the conditions 2, 3, and 4 in Definition 1.1, has been settled, we do not have to resolve it once again in further steps of the procedure. The new grammar G' built as described at steps 1, 2, and 3 has the set of nonterminals N' and the set of productions P'

²This case deals with the possibility of having $Y = B' {}_{B'} A$, too.

composed of all nonterminals from N and productions from P , plus/minus all nonterminals and productions, respectively introduced/removed according to the substitutions performed during the above steps. Next we prove that grammars $G = (N, T, P, S)$ in Chomsky normal form, and $G' = (N', T, P', S)$ in Dyck normal form, generate the same language. Consider the homomorphism $h_d: N' \cup T \rightarrow N \cup T$ defined by $h_d(x) = x$, $x \in T$, $h_d(X) = X$, for $X \in N$, and $h_d(X') = X$ for $X' \in N' - N$, $X \in N$ such that X' is a (transitive³) X' -substitution of X , terminal or not, in the above construction of the grammar G' .

To prove that $L(G') \subseteq L(G)$ we extend h_d to a homomorphism from $(N' \cup T)^*$ to $(N \cup T)^*$ defined on the classical concatenation operation. It is straightforward to prove by induction, that for each $\alpha \Rightarrow_{G'}^* \delta$ we have $h_d(\alpha) \Rightarrow_G^* h_d(\delta)$. This implies that for any derivation of a word $w \in L(G')$, i.e., $S \Rightarrow_{G'}^* w$, we have $h_d(S) \Rightarrow_G^* h_d(w)$, i.e., $S \Rightarrow_G^* w$, or equivalently, $L(G') \subseteq L(G)$.

To prove that $L(G) \subseteq L(G')$ we make use of the CYK (Cocke-Younger-Kasami) algorithm as described in [20]. Let $w = a_1 a_2 \dots a_n$ be an arbitrary word in $L(G)$, and V_{ij} , $i \leq j$, $i, j \in \{1, \dots, n\}$, be the triangular matrix of size $n \times n$ built with the CYK algorithm. Since $w \in L(G)$, we have $S \in V_{1n}$. We prove that $w \in L(G')$, i.e., $S \in V'_{1n}$, where V'_{ij} , $i \leq j$, $i, j \in \{1, \dots, n\}$ forms the triangular matrix obtained by applying the CYK algorithm to w according to G' productions.

We consider two relations $\hat{h}_t \subseteq (N \cup T) \times (N' \cup T)$ and $\hat{h}_{-t} \subseteq N \times N'$. The first relation is defined by $\hat{h}_t(x) = x$, $x \in T$, $\hat{h}_t(S) = S$, if $S \rightarrow t$, $t \in T$, is a rule in G , and $\hat{h}_t(X) = X'$, if X' is a (transitive) X' -terminal substitution⁴ of X , and $X \rightarrow t$ is a rule in G . Finally, $\hat{h}_t(X) = X$ if $X \rightarrow t \in P$, $t \in T$. The second relation is defined as $\hat{h}_{-t}(S) = S$, $\hat{h}_{-t}(X) = \{X\} \cup \{X' | X' \text{ is a (transitive) } X' \text{-nonterminal substitution of } X\}$ and $\hat{h}_{-t}(X) = X$, if there is no substitution of X and no rule of the form $X \rightarrow t$, $t \in T$, in G . Note that $\hat{h}_x(X_1 \cup X_2) = \hat{h}_x(X_1) \cup \hat{h}_x(X_2)$, for $X_i \subseteq N$, $i \in \{1, 2\}$, $x \in \{t, -t\}$. Using \hat{h}_t , each rule $X \rightarrow t$ in P has a corresponding set of rules $\{X' \rightarrow t | X' \in \hat{h}_t(X), X \rightarrow t \in P\}$ in P' . Each rule $A \rightarrow BC$ in P has a corresponding set of rules $\{A' \rightarrow B'C' | A' \in \hat{h}_{-t}(A), B' \in \hat{h}_{-t}(B) \cup \hat{h}_t(B), C' \in \hat{h}_{-t}(C) \cup \hat{h}_t(C), B' \text{ and } C' \text{ are pairwise nonterminals}, A \rightarrow BC \in P\}$ in P' .

Consider $V'_{ii} = \hat{h}_t(V_{ii})$ and $V'_{ij} = \hat{h}_{-t}(V_{ij})$, $i < j$, $i, j \in \{1, \dots, n\}$. We claim that V'_{ij} , $i, j \in \{1, \dots, n\}$, $i \leq j$, defines the triangular matrix obtained by applying CYK algorithm to rules that derive w in G' . First, observe that for $i = j$, we have $V'_{ii} = \hat{h}_t(V_{ii}) = \{A | A \rightarrow a_i \in P'\}$, $i \in \{1, \dots, n\}$, due to the definition of \hat{h}_t . Now let us consider $k = j - i$, $k \in \{1, \dots, n - 1\}$. We want to compute V'_{ij} , $i < j$.

By definition, we have $V_{ij} = \bigcup_{l=i}^{j-1} \{A | A \rightarrow BC, B \in V_{il}, C \in V_{l+1j}\}$, so that $V'_{ij} = \hat{h}_{-t}(V_{ij}) = \hat{h}_{-t}(\bigcup_{l=i}^{j-1} \{A | A \rightarrow BC, B \in V_{il}, C \in V_{l+1j}\}) = \bigcup_{l=i}^{j-1} \hat{h}_{-t}(\{A | A \rightarrow BC, B \in V_{il}, C \in V_{l+1j}\}) = \bigcup_{l=i}^{j-1} \{A' | A' \rightarrow B'C', A' \in \hat{h}_{-t}(A), B' \in \hat{h}_{-t}(B) \cup \hat{h}_t(B), B \in V_{il}, C' \in \hat{h}_{-t}(C) \cup \hat{h}_t(C), C \in V_{l+1j}, B' \text{ and } C' \text{ are pairwise nonterminals}, A \rightarrow BC \in P\}$. Let us explicitly develop the last union.

³There exist $X_k \in N$, such that X' is an X' -substitution of X_k , X_k is an X_k -substitution of X_{k-1}, \dots , and X_1 is an X_1 -substitution of X . All of them substitute X .

⁴There may exist several terminal/nonterminal substitutions for the same nonterminal X . This makes \hat{h}_t/\hat{h}_{-t} to be a relation.

If $k = 1$, then $l \in \{i\}$. For each $i \in \{1, \dots, n-1\}$ we have $V'_{ii+1} = \{A'|A' \rightarrow B'C', A' \in \hat{h}_{-t}(A), B' \in \hat{h}_{-t}(B) \cup \hat{h}_t(B), B \in V_{ii}, C' \in \hat{h}_{-t}(C) \cup \hat{h}_t(C), C \in V_{i+1i+1}, B' \text{ and } C' \text{ are pairwise nonterminals, } A \rightarrow BC \in P\}$. Due to the fact that $B \in V_{ii}$ and $C \in V_{i+1i+1}$, B' is a terminal substitution of B , while C' is a terminal substitution of C . Therefore, we have $B' \notin \hat{h}_{-t}(B)$, $C' \notin \hat{h}_{-t}(C)$, so that $B' \in \hat{h}_t(B)$, for all $B \in V_{ii}$, and $C' \in \hat{h}_t(C)$, for all $C \in V_{i+1i+1}$, i.e., $B' \in \hat{h}_t(V_{ii}) = V'_{ii}$ and $C' \in \hat{h}_t(V_{i+1i+1}) = V'_{i+1i+1}$. Therefore, $V'_{ii+1} = \{A'|A' \rightarrow B'C', B' \in V'_{ii}, C' \in V'_{i+1i+1}\}$.

If $k \geq 2$, then $l \in \{i, i+1, \dots, j-1\}$, and $V'_{ij} = \bigcup_{l=i}^{j-1} \{A'|A' \rightarrow B'C', A' \in \hat{h}_{-t}(A), B' \in \hat{h}_{-t}(B) \cup \hat{h}_t(B), B \in V_{il}, C' \in \hat{h}_{-t}(C) \cup \hat{h}_t(C), C \in V_{l+1j}, B' \text{ and } C' \text{ are pairwise nonterminals, } A \rightarrow BC \in P\}$. We now compute the first set of the above union, i.e., $V'_i = \{A'|A' \rightarrow B'C', A' \in \hat{h}_{-t}(A), B' \in \hat{h}_{-t}(B) \cup \hat{h}_t(B), B \in V_{ii}, C' \in \hat{h}_{-t}(C) \cup \hat{h}_t(C), C \in V_{i+1j}, B' \text{ and } C' \text{ are pairwise nonterminals, } A \rightarrow BC \in P\}$. By the same reasoning as before, the condition $B' \in \hat{h}_{-t}(B) \cup \hat{h}_t(B), B \in V_{ii}$, is equivalent with $B' \in \hat{h}_t(V_{ii}) = V'_{ii}$. Because $i+1 \neq j$, C' is a nonterminal substitution of C . Therefore, $C' \notin \hat{h}_t(C)$, and the condition $C' \in \hat{h}_{-t}(C) \cup \hat{h}_t(C), C \in V_{i+1j}$ is equivalent with $C' \in \hat{h}_{-t}(V_{i+1j}) = V'_{i+1j}$. So that $V'_i = \{A'|A' \rightarrow B'C', B' \in V'_{ii}, C' \in V'_{i+1j}\}$. Using the same method for each $l \in \{i+1, \dots, j-1\}$ we have $V'_l = \{A'|A' \rightarrow B'C', A' \in \hat{h}_{-t}(A), B' \in \hat{h}_{-t}(B) \cup \hat{h}_t(B), B \in V_{il}, C' \in \hat{h}_{-t}(C) \cup \hat{h}_t(C), C \in V_{l+1j}, B' \text{ and } C' \text{ are pairwise nonterminals, } A \rightarrow BC \in P\} = \{A'|A' \rightarrow B'C', B' \in V'_{il}, C' \in V'_{l+1j}\}$. In conclusion, $V'_{ij} = \bigcup_{l=i}^{j-1} \{A'|A' \rightarrow B'C', B' \in V'_{il}, C' \in V'_{l+1j}\}$, for each $i, j \in \{1, \dots, n\}$, i.e., V'_{ij} , $i \leq j$, contains the nonterminals of the $n \times n$ triangular matrix computed by applying the CYK algorithm to rules that derive w in G' . Because $w \in L(G)$, we have $S \in V_{1n}$. That is equivalent with $S \in V'_{1n} = \hat{h}_t(V_{1n})$, if $n = 1$, and $S \in V'_{1n} = \hat{h}_{-t}(V_{1n})$, if $n > 1$, i.e., $w \in L(G')$. \square

Corollary 1.3. *Let G be a context-free grammar in Dyck normal form. Any terminal derivation in G producing a word of length n , $n \geq 1$, takes $2n - 1$ steps.*

Proof. If G is a context-free grammar in Dyck normal form, then it is also in Chomsky normal form, and all properties of the latter hold. \square

Corollary 1.4. *If $G = (N, T, P, S)$ is a grammar in Chomsky normal form, and $G' = (N', T, P', S)$ its equivalent in Dyck normal form, then there exists a homomorphism $h_d: N' \cup T \rightarrow N \cup T$, such that any derivation tree of $w \in L(G)$ is the homomorphic image of a derivation tree of the same word in G' .*

Proof. Consider the homomorphism $h_d: N' \cup T \rightarrow N \cup T$ defined as $h_d(A_t) = h_d(zA) = h_d(A_z) = A$, for each A_t -terminal or $zA(A_z)$ -nonterminal substitution of A , and $h_d(t) = t$, $t \in T$. The claim is a direct consequence of the way in which the new nonterminals A_t , zA , and A_z have been chosen. \square

Note that, due to the pairwise-renaming procedure used to reach the Dyck normal form, it may appear that a context-free grammar in Dyck normal form is more ambiguous than the original grammar in Chomsky normal form. However, this is relative. The derivation trees of a certain word have the same structure in both grammars, in Chomsky normal form

and Dyck normal form (only some “labels” of the nodes in these trees differ). The apparent ambiguity can be resolved through the homomorphism h_d considered in Corollary 1.4.

Let G be a grammar in Dyck normal form. To emphasize the pairwise brackets occurring on the right-hand side of a rule, and also to make the connection with the Dyck language, each pair (A, B) , such that there exists a rule of the form $X \rightarrow AB$, is replaced by an indexed pair of brackets $[_i,]_i$. In each rule that rewrites A and B , we replace A by $[_i$, and B by $]_i$, respectively. Next we present an example of the conversion procedure described in the proof of Theorem 1.2 along with the homomorphism considered in Corollary 1.4.

Example 1.5. Consider the context-free grammar in Chomsky normal form $G = (\{E_0, E, E_1, E_2, T, T_1, T_2, R\}, \{+, *, a\}, E_0, P')$, where $P' = \{E_0 \rightarrow a/TT_1/EE_1, E \rightarrow a/TT_1/EE_1, T \rightarrow a/TT_1, T_1 \rightarrow T_2R, E_1 \rightarrow E_2T, T_2 \rightarrow *, E_2 \rightarrow +, R \rightarrow a\}$.

To convert G into Dyck normal form, with respect to Definition 1.1, item 2, we first remove $E \rightarrow a$ and $T \rightarrow a$. Then, according to item 3, we remove the right occurrence of T from the rule $E_1 \rightarrow E_2T$, along with other transformations that may be required after completing these procedures. Let E_3 and T_3 be two new nonterminals. We remove $E \rightarrow a$ and $T \rightarrow a$, and add the rules $E_3 \rightarrow a$, $T_3 \rightarrow a$, $E_0 \rightarrow E_3E_1$, $E_0 \rightarrow T_3T_1$, $E \rightarrow E_3E_1$, $E \rightarrow T_3T_1$, $E_1 \rightarrow E_2T_3$, $T \rightarrow T_3T_1$. Let T' be the new nonterminal that replaces the right occurrence of T . We add the rules $E_1 \rightarrow E_2T'$, $T' \rightarrow TT_1$, $T' \rightarrow T_3T_1$, and remove $E_1 \rightarrow E_2T$. We repeat the procedure with T_3 (added in the previous step), i.e., we introduce a new nonterminal T_4 , remove $E_1 \rightarrow E_2T_3$, add $E_1 \rightarrow E_2T_4$ and $T_4 \rightarrow a$.

Due to the new nonterminals E_3, T_3, T_4 , item 4 does not hold. To have accomplished this condition, we introduce three new nonterminals E_4 to replace E_2 in $E_1 \rightarrow E_2T_4$, E_5 to replace E_1 in $E_0 \rightarrow E_3E_1$ and $E \rightarrow E_3E_1$, and T_5 to replace T_1 in $E_0 \rightarrow T_3T_1$ and $E \rightarrow T_3T_1$. We remove all the above rules and add the new rules $E_1 \rightarrow E_4T_4$, $E_4 \rightarrow +$, $E_0 \rightarrow E_3E_5$, $E \rightarrow E_3E_5$, $E_5 \rightarrow E_2T'$, $E_5 \rightarrow E_4T_4$, $E_0 \rightarrow T_3T_5$, $E \rightarrow T_3T_5$, and $T_5 \rightarrow T_2R$.

The Dyck normal form of G' , in bracketed notation, is $G'' = (\{E_0, [_1, [_2, \dots, [_7,]_1,]_2, \dots,]_7\}, \{+, *, a\}, E_0, P'')$, $P'' = \{E_0 \rightarrow a/[_1]_1/[_2]_2/[_3]_3/[_4]_4, [_1 \rightarrow [_1]_1/[_4]_4, [_2 \rightarrow [_1]_1/[_2]_2/[_3]_3/[_4]_4, [_1 \rightarrow [_7]_7, [_2 \rightarrow [_5]_5/[_6]_6, [_3 \rightarrow [_5]_5/[_6]_6, [_4 \rightarrow [_7]_7, [_5 \rightarrow [_1]_1/[_4]_4, [_3 \rightarrow a, [_4 \rightarrow a, [_5 \rightarrow +, [_6 \rightarrow +,]_6 \rightarrow a,]_7 \rightarrow *,]_7 \rightarrow a\}$, where $([T,]T_1) = ([_1,]_1)$, $([E,]E_1) = ([_2,]_2)$, $([E_3,]E_5) = ([_3,]_3)$, $([T_3,]T_5) = ([_4,]_4)$, $([E_2,]T') = ([_5,]_5)$, $([E_4,]T_4) = ([_6,]_6)$, $([T_2,]R) = ([_7,]_7)$.

The homomorphism h_d is defined as $h_d: N' \cup T \rightarrow N'' \cup T$, $h_d(E_0) = E_0$, $h_d([_2) = h_d([_3) = E$, $h_d([_2) = h_d([_3) = E_1$, $h_d([_5) = h_d([_6) = E_2$, $h_d([_1) = h_d([_5) = h_d([_4) = h_d([_6) = T$, $h_d([_1) = h_d([_4) = T_1$, $h_d([_7) = T_2$, $h_d([_7) = R$, $h_d(t) = t$, for each $t \in T$.

The string $w = a * a * a + a$ is a word in $L(G'') = L(G)$ generated, for instance, by a leftmost derivation D in G'' as follows.

$D: E_0 \Rightarrow [_2]_2 \Rightarrow [_1]_1]_2 \Rightarrow [_4]_4]_1]_2 \Rightarrow a]_4]_1]_2 \Rightarrow a [_7]_7]_1]_2 \Rightarrow a *]_7]_1]_2 \Rightarrow a *]_1]_2 \Rightarrow a * a [_7]_7]_2 \Rightarrow a * a *]_7]_2 \Rightarrow a * a * a]_2 \Rightarrow a * a * a [_6]_6 \Rightarrow a * a * a +]_6 \Rightarrow a * a * a + a$.

Applying h_d to D , in G' , we obtain a derivation of w in G' . If we consider \mathcal{T} the derivation tree of w in G , and \mathcal{T}' the derivation tree of w in G'' , then \mathcal{T} is the homomorphic image of \mathcal{T}' through h_d .

2 Characterizations of Context-Free Languages by Dyck Languages

Definition 2.1. Let $G_k = (N_k, T, P_k, S)$ be a context-free grammar in Dyck normal form with $|N_k - \{S\}| = 2k$. Let $D: S \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_{2n-1} = w$, $n \geq 2$, be a leftmost derivation of $w \in L(G)$. The *trace-word* of w associated with the derivation D , denoted as $t_{w,D}$, is defined as the concatenation of nonterminals consecutively rewritten in D , excluding the axiom. The *trace-language* associated with G_k , denoted by $L(G_k)$, is $L(G_k) = \{t_{w,D} \mid \text{for any } w \in L(G_k), \text{ and any leftmost derivation } D \text{ of } w\}$.

Note that $t_{w,D}$, $w \in L(G)$, can also be read from the derivation tree in the depth-first search order starting with the root, but ignoring the root and the leaves. The trace-word associated with w and the leftmost derivation D in Example 2.5 is $t_{a*a*a+a,D} = [E [T [T_3]T_5 [T_2]R]T_1 [T_2]R]E_1 [E_4]T_4$.

Definition 2.2. A *one-sided Dyck language* over k letters, $k \geq 1$, is a context-free language defined by the grammar $\Gamma_k = (\{S\}, T_k, P, S)$, where $T_k = \{[1, [2, \dots, [k,]_1,]_2, \dots,]_k\}$ and $P = \{S \rightarrow [i S]_i, S \rightarrow SS, S \rightarrow [i]_i \mid 1 \leq i \leq k\}$.

Let $G_k = (N_k, T, P_k, S)$ be a context-free grammar in Dyck normal form. To emphasize possible relations between the structure of trace-words in $L(G_k)$ and the structure of words in the Dyck language, and also to keep control of each bracketed pair occurring on the right-hand side of each rule in G_k , we fix $N_k = \{S, [1, [2, \dots, [k,]_1,]_2, \dots,]_k\}$, and P_k to be composed of rules of the forms $X \rightarrow [i]_i$, $1 \leq i \leq k$, and $Y \rightarrow t$, $X, Y \in N_k$, $t \in T$. From [19] we have adopted the next characterizations of D_k , $k \geq 1$, (Definition 2.3, and Lemmas 2.4 and 2.5).

Definition 2.3. For a string w , let $w_{i:j}$ be its substring starting at the i^{th} position and ending at the j^{th} position. Let h be a homomorphism defined as follows:

$$h([1] = h([2] = \dots = h([k] = [1, \quad h(]1) = h(]2) = \dots = h(]k) =]_1.$$

Let $w \in D_k$, $1 \leq i \leq j \leq |w|$, where $|w|$ is the length of w . We say that (i, j) is a *matched pair* of w , if $h(w_{i:j})$ is *balanced*, i.e., $h(w_{i:j})$ has an equal number of $[1$'s and $]_1$'s and, in any prefix of $h(w_{i:j})$, the number of $[1$'s is greater than or equal to the number of $]_1$'s.

Lemma 2.4. A string $w \in \{[1,]_1\}^*$ is in D_1 if and only if it is balanced.

Consider the homomorphisms defined as follows (where λ is the empty string)

$$\begin{aligned} h_1([1] &= [1, \quad h_1(]1) =]_1, \quad h_1([2] = h_1(]2) = \dots = h_1([k] = h_1(]k) = \lambda, \\ h_2([2] &= [1, \quad h_2(]2) =]_1, \quad h_2([1] = h_2(]1) = \dots = h_2([k] = h_2(]k) = \lambda, \dots \\ h_k([k] &= [1, \quad h_k(]k) =]_1, \quad h_k([1] = h_k(]1) = \dots = h_k([k-1] = h_k(]k-1) = \lambda. \end{aligned}$$

Lemma 2.5. We have $w \in D_k$, $k \geq 2$, if and only if the following conditions hold: i) $(1, |w|)$ is a matched pair, and ii) for all matched pairs (i, j) , $h_k(w_{i:j})$ are in D_1 , where $k \geq 1$.

Definition 2.6. Let $w \in D_k$, (i, j) is a *nested pair* of w if (i, j) is a matched pair, and either $j = i + 1$, or $(i + 1, j - 1)$ is a matched pair.

Definition 2.7. Let $w \in D_k$ and (i, j) be a matched pair of w . We say that (i, j) is *reducible* if there exists an integer j' , $i < j' < j$, such that (i, j') and $(j' + 1, j)$ are matched pairs.

Let $w \in D_k$, if (i, j) is a nested pair of w then (i, j) is an irreducible pair. If (i, j) is a nested pair of w then $(i + 1, j - 1)$ may be a reducible pair.

Theorem 2.8. *The trace-language associated with a context-free grammar, $G = (N_k, T, P_k, S)$ in Dyck normal form, with $|N_k| = 2k + 1$, is a subset of D_k .*

Proof. Let $N_k = \{S, [1, \dots, [k,]_1, \dots,]_k\}$ be the set of nonterminals, $w \in L(G)$, and D a leftmost derivation of w . We show that any subtree of the derivation tree, read in the depth-first search order, by ignoring the root and the terminal nodes, corresponds to a matched pair in $t_{w,D}$. In particular, $(1, |t_{w,D}|)$ will be a matched pair. Denote by $t_{w,D_{i:j}}$ the substring of $t_{w,D}$ starting at the i^{th} position and ending at the j^{th} position of $t_{w,D}$. We show that for all matched pairs (i, j) , $h_{k'}(t_{w,D_{i:j}})$ belong to D_1 , $1 \leq k' \leq k$. We prove these claims by induction on the height of subtrees.

Basis. Certainly, any subtree of height $n = 1$, read in the depth-first search order, looks like $[i]_i$, $1 \leq i \leq k$. Therefore, it satisfies the above conditions.

Induction step. Assume that the claim is true for all subtrees of height \bar{h} , $\bar{h} < n$, and we prove it for $\bar{h} = n$. Each subtree of height n can have one of the following structures. The level 0 of the subtree is marked by a left or right bracket. This bracket will not be considered when we read the subtree. Denote by $[_m$ the left son of the root. Then the right son is labeled by $]_m$. They are the roots of a left and right subtree, for which at least one has the height $n - 1$.

Suppose that both subtrees have the height $1 \leq \bar{h} \leq n - 1$. By the induction hypothesis, let us further suppose that the left subtree corresponds to the matched pair (i_l, j_l) , and the right subtree corresponds to the matched pair (i_r, j_r) , $i_r = j_l + 2$, because the position $j_l + 1$ is taken by $]_m$. As h is a homomorphism, we have $h(t_{w,D_{i_l-1:j_r}}) = h([_m t_{w,D_{i_l:j_l}}]_m t_{w,D_{j_l+2:j_r}}) = h([_m)h(t_{w,D_{i_l:j_l}})h(]_m)h(t_{w,D_{j_l+2:j_r}})$. Therefore, $h(t_{w,D_{i_l-1:j_r}})$ satisfies all conditions in Definition 2.3, and thus $(i_l - 1, j_r)$ that corresponds to the considered subtree of height n , is a matched pair. By the induction hypothesis, $h_{k'}(t_{w,D_{i_l:j_l}})$ and $h_{k'}(t_{w,D_{i_r:j_r}})$ are in D_1 , $1 \leq k' \leq k$. Hence, $h_{k'}(t_{w,D_{i_l-1:j_r}}) = h_{k'}([_m)h_{k'}(t_{w,D_{i_l:j_l}})h_{k'}(]_m)h_{k'}(t_{w,D_{j_l+2:j_r}}) \in \{h_{k'}(t_{w,D_{i_l:j_l}})h_{k'}(t_{w,D_{j_l+2:j_r}}), [1h_{k'}(t_{w,D_{i_l:j_l}})]_1h_{k'}(t_{w,D_{j_l+2:j_r}})\}$ belong to D_1 , $1 \leq k' \leq k$. Note that in this case the matched pair $(i_l - 1, j_r)$ is reducible into $(i_l - 1, j_l + 1)$ and $(j_l + 2, j_r)$, where $(i_l - 1, j_l + 1)$ corresponds to the substring $t_{w,D_{i_l-1:j_l+1}} = [_m t_{w,D_{i_l:j_l}}]_m$. We refer to this structure as the *left embedded subtree*, i.e., $(i_l - 1, j_l + 1)$ is a nested pair. A similar reasoning is applied for the case when one of the subtrees has the height 0. Analogously, it can be shown that the initial tree corresponds to the matched pair $(1, |t_{w,D}|)$, i.e., the first condition of Lemma 2.5 holds. So far, we have proved that each subtree of the derivation tree, and also each left embedded subtree, corresponds to a matched pair (i, j) and (i_l, j_l) , such that $h_{k'}(t_{w,D_{i:j}})$ and $h_{k'}([_m t_{w,D_{i_l:j_l}}]_m)$, $1 \leq k' \leq k$, are in D_1 .

Next we show that all matched pairs from $t_{w,D}$ correspond only to subtrees, or left embedded subtrees, from the derivation tree. To derive a contradiction, let us suppose that there exists a matched pair (i, j) in $t_{w,D}$, that does not correspond to any subtree, or left

embedded subtree, of the derivation tree read in the depth-first search order. We show that this leads to a contradiction.

Since (i, j) does not correspond to any subtree, or left embedded subtree, there exist two adjacent subtrees θ_1 (a left embedded subtree) and θ_2 (a right subtree) such that (i, j) is composed of two adjacent “subparts” of θ_1 and θ_2 . In terms of matched pairs, if θ_1 corresponds to the matched pair (i_1, j_1) and θ_2 corresponds to the matched pair (i_2, j_2) , such that $i_2 = j_1 + 2$, then there exists a suffix $s_{i_1-1:j_1+1}$ of $t_{w,D_{i_1-1:j_1+1}}$, and a prefix $p_{i_2:j_2}$ of $t_{w,D_{i_2:j_2}}$, such that $t_{w,D_{i:j}} = s_{i_1-1:j_1+1}p_{i_2:j_2}$. Furthermore, without loss of generality, we assume that (i_1, j_1) and (i_2, j_2) are nested pairs. Otherwise, the matched pair (i, j) can be “narrowed” until θ_1 and θ_2 are characterized by two nested pairs. If (i_1, j_1) is a nested pair, then so is $(i_1 - 1, j_1 + 1)$. As $s_{i_1-1:j_1+1}$ is a suffix of $t_{w,D_{i_1-1:j_1+1}}$ and $(i_1 - 1, j_1 + 1)$ is a matched pair, with respect to Definition 2.3, the number of $]_1$ ’s in $h(s_{i_1-1:j_1+1})$ is greater than or equal to the number of $[_1$ ’s in $h(s_{i_1-1:j_1+1})$. On the other hand, $s_{i_1-1:j_1+1}$ is also a prefix of $t_{w,D_{i:j}}$, because (i, j) is a matched pair, by the induction hypothesis. Therefore, the number of $[_1$ ’s in $h(s_{i_1-1:j_1+1})$ is greater than or equal to the number of $]_1$ ’s in $h(s_{i_1-1:j_1+1})$. Hence, the only possibility for $s_{i_1-1:j_1+1}$ to be, in the same time, a suffix for $t_{w,D_{i_1-1:j_1+1}}$ and a prefix for $t_{w,D_{i:j}}$, is the equality between the number of $[_1$ ’s and $]_1$ ’s in $h(s_{i_1-1:j_1+1})$. This property holds if and only if $s_{i_1-1:j_1+1}$ corresponds to a matched pair in $t_{w,D_{i_1-1:j_1+1}}$, i.e., if i_s and j_s are the start and the end positions of $s_{i_1-1:j_1+1}$ in $t_{w,D_{i_1-1:j_1+1}}$, then (i_s, j_s) is a matched pair. Thus, $(i_1 - 1, j_1 + 1)$ is a reducible pair into $(i_1 - 1, i_s - 1)$ and (i_s, j_s) , where $j_s = j_1 + 1$. We have reached a contradiction, i.e., $(i_1 - 1, j_1 + 1)$ is reducible.

Therefore, the matched pairs in $t_{w,D}$ correspond to subtrees, or left embedded subtrees, in the derivation tree. For these matched pairs we have already proved that they satisfy Lemma 2.5. Accordingly, $t_{w,D} \in D_k$, and consequently the trace-language associated with G is a subset of D_k . \square

Theorem 2.9. *Given a context-free grammar G there exist an integer K , a homomorphism φ , and a subset D'_K of the Dyck language D_K , such that $L(G) = \varphi(D'_K)$.*

Proof. Let G be a context-free grammar and $G_k = (N_k, T, P_k, S)$ be the Dyck normal form of G , such that $N_k = \{S, [1, \dots, [k,]_1, \dots,]_k\}$. Let $L(G_k)$ be the trace-language associated with G_k . Consider $\{t_{k+1}, \dots, t_{k+p}\}$ the ordered subset of T , such that $S \rightarrow t_{k+i} \in P$, $1 \leq i \leq p$. We define $N_{k+p} = N_k \cup \{[t_{k+1}, \dots, [t_{k+p},]_{t_{k+1}}, \dots,]_{t_{k+p}}\}$, and $P_{k+p} = P_k \cup \{S \rightarrow [t_{k+i}]_{t_{k+i}}, [t_{k+i} \rightarrow t_{k+i},]_{t_{k+i}} \rightarrow \lambda \mid S \rightarrow t_{k+i} \in P, 1 \leq i \leq p\}$. The new grammar $G_{k+p} = (N_{k+p}, T, P_{k+p}, S)$ generates the same language as G_k .

Let $\varphi: (N_{k+p} - \{S\})^* \rightarrow T^*$ be the homomorphism defined by $\varphi(N) = \lambda$, for each rule of the form $N \rightarrow XY$, $N, X, Y \in N_k - \{S\}$, and $\varphi(N) = t$, for each rule of the form $N \rightarrow t$, $N \in N_k - \{S\}$, and $t \in T$, $\varphi([_{k+i}) = t_{k+i}$, and $\varphi(]_{k+i}) = \lambda$, for each $1 \leq i \leq p$. Obviously, $L = \varphi(D'_K)$, where $K = k + p$, $D'_K = L(G_k) \cup L_p$, and $L_p = \{[t_{k+1}]_{t_{k+1}}, \dots, [t_{k+p}]_{t_{k+p}}\}$. \square

In the sequel, grammar G_{k+p} is called the *extended grammar* of G_k . G_k has an extended grammar if and only if G_k (or G) has rules of the form $S \rightarrow t$, $t \in T \cup \{\lambda\}$. If G_k does not have an extended grammar then $D'_K = D'_k = L(G_k)$.

3 On the Chomsky-Schützenberger Theorem

Let $G_k = (N_k, T, P_k, S)$ be an arbitrary context-free grammar in Dyck normal form, with $N_k = \{S, [1, \dots, [k,]_1, \dots,]_k\}$. and $\varphi: (N_k - \{S\})^* \rightarrow T^*$ the restriction of the homomorphism φ in the proof of Theorem 2.9. We divide N_k into three main sets $N^{(1)}$, $N^{(2)}$, $N^{(3)}$ as follows:

1. $[_i$ and $]_i$ belong to $N^{(1)}$ if and only if $\varphi([_i) = t$ and $\varphi(]_i) = t'$, $t, t' \in T$,
2. $[_i$ and $]_i$ belong to $N^{(2)}$ if and only if $\varphi([_i) = t$ and $\varphi(]_i) = \lambda$, or vice versa $\varphi([_i) = \lambda$ and $\varphi(]_i) = t$, $t \in T$,
3. $[_i,]_i \in N^{(3)}$ if and only if $\varphi([_i) = \lambda$ and $\varphi(]_i) = \lambda$.

Certainly, $N_k - \{S\} = N^{(1)} \cup N^{(2)} \cup N^{(3)}$ and $N^{(1)} \cap N^{(2)} \cap N^{(3)} = \emptyset$. $N^{(2)}$ is further divided into $N_l^{(2)}$ and $N_r^{(2)}$, where $N_l^{(2)}$ contains those pairs $[_i,]_i \in N^{(2)}$ such that $\varphi([_i) \neq \lambda$, while $N_r^{(2)}$ contains those pairs $[_i,]_i \in N^{(2)}$ such that $\varphi(]_i) \neq \lambda$.

Definition 3.1. A grammar G_k is in linear-Dyck normal form if G_k is in Dyck normal form and $N^{(3)} = \emptyset$.

Theorem 3.2. For each linear grammar G , there exists a grammar G_k in linear-Dyck normal form such that $L(G) = L(G_k)$, and vice versa.

Proof. Each linear grammar G , in standard form, is composed of rules of the forms $X \rightarrow \lambda$, $X \rightarrow t$, $X \rightarrow t_1 Y$, $X \rightarrow Y t_2$, $X \rightarrow t_1 Y t_2$, $t, t_1, t_2 \in T$, $X, Y \in N$. Transforming G into Chomsky normal form, and then into the Dyck normal form, we obtain a grammar G_k in linear-Dyck normal form. Since the standard form for linear languages, Chomsky normal form, and Dyck normal form are weakly equivalent we obtain $L(G) = L(G_k)$. The converse statement is trivial. \square

Next we consider more closely the structures of the derivation trees associated with words generated by linear and context-free grammars in linear-Dyck normal form and Dyck normal form, respectively. We are interested on the structure of the trace-words associated with words generated by these grammars.

Let $G_k = (N_k, T, P_k, S)$ be an arbitrary (linear) context-free grammar in (linear-)Dyck normal form, and $L(G_k)$ the language generated by this grammar. Let $w \in L(G_k)$, D a leftmost derivation of w , and $t_{w,D}$ the trace-word of w associated with D . From the structure of the derivation tree, read in the depth-first search order, it is easy to observe that each bracket $[_i$, such that $[_i,]_i \in N^{(1)}$, is immediately followed, in $t_{w,D}$ by its pairwise $]_i$. The same property holds for those pairs $[_i,]_i \in N_l^{(2)}$. If $[_i,]_i \in N_r^{(2)} \cup N^{(3)}$ then the pair $[_i,]_i$ should embed a left subtree, i.e., the case of the left embedded subtree in the proof of Theorem 2.8. In this case the bracket $[_i$ may have a left, long distance, placement from its pairwise $]_i$, in $t_{w,D}$.

Suppose that G_k is a linear grammar in linear-Dyck normal form, i.e., $N^{(3)} = \emptyset$, such that $N_l^{(2)} \neq \emptyset$ and $N_r^{(2)} \neq \emptyset$. Each word $w = a_1 a_2 \dots a_n \in L(G_k)$, of an arbitrary length n , has the property that there exists an index n_t , $1 \leq n_t \leq n - 1$, and a unique pair⁵

⁵To emphasize which of the brackets in the pair $([_i,]_i)$ produces a terminal, we also use the notation $[_i^t,]_i^t$ if and only if $[_i,]_i \in N_r^{(2)}$, $[_i^t,]_i$ if and only if $[_i,]_i \in N_l^{(2)}$, and $[_i^t,]_i^t$ if and only if $[_i,]_i \in N^{(1)}$.

of the loop complexity formally defined⁶ and studied in [3] and [7]. In [7] it is proved that from each two vertices u and v belonging to a digraph of cycle rank k , there exists a regular expression of star-height⁷ at most k that describes the set of paths from u to v . On the other hand, the cycle rank of a digraph with n vertices is upper bounded by $n \log n$ [13]. Hence any regular expression obtained from a digraph with n vertices has the star-height at most $n \log n$. Consequently, the (infinite) set of paths from an initial vertex to a final vertex in \mathcal{G}^X , can be divided into a finite number of classes of terminal paths. Paths belonging to the same class are characterized by the same regular expression, in terms of $*$ and $+$ Kleene operations, of star-height at most $|V_X| \log |V_X|$ (which is finite related to the lengths of strings in $L(G_k)$).

Denote by $\mathcal{R}_{[i]}^X$ the set of all regular expressions over $\tilde{N}_k \cup \{X\}$ that can be read in \mathcal{G}^X , starting from the initial vertex X and ending in the final vertex $[i]$. The cardinality of $\mathcal{R}_{[i]}^X$ is finite. Define the homomorphism $h_{\mathcal{G}}: \tilde{N}_k \cup \{X\} \rightarrow \{[i] | [i] \in N_r^{(2)} \cup N^{(3)}\} \cup \{\lambda\}$ such that $h_{\mathcal{G}}([i] = [i]$ for any $[i] \in N_r^{(2)} \cup N^{(3)}$, $h_{\mathcal{G}}(X) = h_{\mathcal{G}}([i] = [i] = \lambda$, for any $[i] \in N^{(1)}$ and $[i] \in N_l^{(2)}$. For any element $r.e_{[i]}^{(l,X)} \in \mathcal{R}_{[i]}^X$ we build a new regular expression⁸ $r.e_{[i]}^{(r,X)} = h_{\mathcal{G}}^r(r.e_{[i]}^{(l,X)})$, where $h_{\mathcal{G}}^r$ is the mirror image of $h_{\mathcal{G}}$. Consider $r.e_{[i]}^X = r.e_{[i]}^{(l,X)} r.e_{[i]}^{(r,X)}$. For a certain X and $[i]$, denote by $\mathcal{R}.e_{[i]}^X$ the set of all regular expressions $r.e_{[i]}^X$ obtained as above. Furthermore, $\mathcal{R}.e^X = \bigcup_{[i], [i] \in N^{(1)}} \mathcal{R}.e_{[i]}^X$ and $\mathcal{R}.e = \mathcal{R}.e^S \cup (\bigcup_{[i], [i] \in N^{(3)}} \mathcal{R}.e^{[i]})$.

Construction 3.4. Let $G_k = (N_k, T, P_k, S)$ be a context-free grammar in Dyck normal form and $\{\mathcal{G}^X | X \in \{[j] | [j] \in N^{(3)} \cup \{S\}\}\}$ the set of dependency graphs of G_k . The *extended dependency graph* of G_k , denoted by $\mathcal{G}_e = (\mathcal{V}_e, \mathcal{E}_e)$, is a directed graph for which $\mathcal{V}_e = \tilde{N}_k \cup \{S\} \cup \{[i] | [i] \in N_r^{(2)} \cup N^{(3)}\}$, S is the initial vertex of \mathcal{G}_e and \mathcal{E}_e is built as follows:

1. - $S[i]$ ($S[j]$) - there exists an edge in \mathcal{G}_e from the vertex labeled by S to a vertex labeled by $[i]$ (from S to $[j]$), $[i], [i] \in N^{(1)} \cup N_r^{(2)} \cup N^{(3)}$ ($[j], [j] \in N_l^{(2)}$), if there exists a regular expression in $\mathcal{R}.e^S$ with a prefix of the form $S[i]$ ($S[j]$, respectively).
2. - $]i[j]$ - there exists an edge in \mathcal{G}_e from a vertex labeled by $]i$ to a vertex labeled by $]j$, $[i], [i], [j], [j] \in N_l^{(2)}$, if there exists a regular expression in $\mathcal{R}.e$ having a substring of the form $]i[j]$ (if $i = j$ then $]i[i]$ forms a self-loop in \mathcal{G}_e).
3. - $]i[j]$ (or $]j[i]$) - there exists an edge in \mathcal{G}_e from a vertex labeled by $]i$ to a vertex labeled by $]j$ (or vice versa from $]j$ to $]i$) such that $[i], [i] \in N_l^{(2)}$ and $[j], [j] \in N_r^{(2)} \cup N^{(3)}$, if there exists a regular expression in $\mathcal{R}.e$ having a substring of the form $]i[j]$ ($]j[i]$, respectively).
4. - $]i[j]$ - there exists an edge in \mathcal{G}_e from a vertex labeled by $]i$ to a vertex labeled by $]j$, $[i], [i], [j], [j] \in N_r^{(2)} \cup N^{(3)}$, if there exists a regular expression in $\mathcal{R}.e$ having a substring of the

⁶In brief, the rank of a cycle \mathcal{C} is 1 if there exists $v \in \mathcal{C}$ such that $\mathcal{C} - v$ is not a cycle. Recursively, the rank of a cycle \mathcal{C} is k if there exists $v \in \mathcal{C}$ such that $\mathcal{C} - v$ contain a cycle of rank $k - 1$ and all the other cycles in $\mathcal{C} - v$ have the rank at most $k - 1$.

⁷Informally, this is the (maximal) power of a nested $*$ -loop occurring in the description of a regular expression. For the formal definition the reader is referred to [7] and [18] (see also Definition 4.1, Section 4).

⁸Since regular languages are closed under homomorphism and reverse operation, $r.e_{[i]}^{(r,X)}$ is a regular expression.

form $]_i[_j$ (if $i = j$ then $]_i[_i$ forms a self-loop in \mathcal{G}_e).

5. - $]_i[_j^t$ (or $]_i[_i^t$) - there exists an edge in \mathcal{G}_e from a vertex labeled by $]_i$ (or by $]_i$) to a vertex labeled by $[_j^t$, $]_i, [_j \in N_l^{(2)}$ (or $]_i, [_j \in N_r^{(2)}$, respectively), $[_j^t,]_i^t \in N^{(1)}$, if there exists a regular expression in $\mathcal{R}.e$ with a substring of the form $]_i[_j^t$ ($]_i[_i^t$, respectively).

6. - $]_j[_i^t$ - there exists an edge in \mathcal{G}_e from a vertex labeled by $]_j$ to a vertex labeled by $[_i^t$, $]_j, [_i \in N^{(3)}$, $[_i^t,]_j^t \in N^{(1)}$, if there exists a regular expression in $\mathcal{R}.e^j_{[_i^t}$ of the form $]_j[_i^t$.

7. - $]_j[_i$ (or $]_j[_j$) - there exists an edge in \mathcal{G}_e from a vertex labeled by $]_j$ to a vertex labeled by $]_i$, $]_j, [_i \in N^{(3)}$, $]_j, [_i \in N_r^{(2)}$ ($]_j, [_i \in N_l^{(2)}$, respectively), if there exists a regular expression in $\mathcal{R}.e^j$ with a prefix of the form $]_j[_i$ ($]_j[_j$, respectively).

8. - $]_i[_j$ - there exists an edge in \mathcal{G}_e from a vertex $]_i$ to a vertex labeled by $]_j$, i and j not necessarily distinct, such that $]_i, [_i \in N_r^{(2)}$, $]_j, [_j \in N_r^{(2)} \cup N^{(3)}$, if either *i.*, *ii.*, or *iii.* holds:

i. there exists at least one regular expression in $\mathcal{R}.e$ having a substring of the form $]_i[_j$ (if $i = j$ then $]_i[_i$ forms a self-loop in \mathcal{G}_e),

ii. there exists $]_k[_k \in N^{(3)}$ such that there exist a regular expression in $\mathcal{R}.e$ with a substring of the form $]_k[_j$, and a regular expression in $\mathcal{R}.e^{]_k}$ that ends in $]_i$ (if $i = j$ then $]_i[_i$ is a self-loop).

iii. there exist $]_k[_k,]_{k_1}[_{k_1}, \dots,]_{k_m}[_{k_m} \in N^{(3)}$ such that there exist a regular expression in $\mathcal{R}.e$ with a substring of the form $]_k[_j$, a regular expression in $\mathcal{R}.e^{]_k}$ that ends in $]_{k_1}$, a regular expression in $\mathcal{R}.e^{]_{k_1}}$ that ends in $]_{k_2}$, and so on, until a regular expression in $\mathcal{R}.e^{]_{k_{m-1}}}$ ending in $]_{k_m}$ and a regular expression in $\mathcal{R}.e^{]_{k_m}}$ ending in $]_i$ are reached.

9. - $[_i^t[_j$ - there exists an edge in \mathcal{G}_e from a vertex labeled by $[_i^t$, $[_i^t,]_i^t \in N^{(1)}$, to a vertex labeled by $]_j$, $]_j, [_j \in N_r^{(2)} \cup N^{(3)}$ if either *i.*, *ii.*, or *iii.* holds

i. there exists a regular expression in $\mathcal{R}.e$ having a substring of the form $[_i^t[_j$,

ii. there exists $]_k[_k \in N^{(3)}$ such that there exist a regular expression in $\mathcal{R}.e$ having a substring of the form $]_k[_j$, and a regular expression in $\mathcal{R}.e^{]_k}_{[_i^t}$ that ends in $[_i^t$.

iii. there exist $]_k[_k,]_{k_1}[_{k_1}, \dots,]_{k_m}[_{k_m} \in N^{(3)}$ such that there exist a regular expression in $\mathcal{R}.e$ with a substring of the form $]_k[_j$, a regular expression in $\mathcal{R}.e^{]_k}$ that ends in $]_{k_1}$, a regular expression in $\mathcal{R}.e^{]_{k_1}}$ that ends in $]_{k_2}$, and so on, until a regular expression in $\mathcal{R}.e^{]_{k_{m-1}}}$ ending in $]_{k_m}$ and a regular expression in $\mathcal{R}.e^{]_{k_m}}_{[_i^t}$ ending in $[_i^t$ are reached.

10. - A vertex labeled by $[_i^t$, $[_i^t,]_i^t \in N^{(1)}$, is a final vertex in \mathcal{G}_e if either *i.*, *ii.*, or *iii.* holds:

i. there exists a regular expression in $\mathcal{R}.e^S$ that ends in $[_i^t$,

ii. there exists $]_k[_k \in N^{(3)}$, such that there exist a regular expression in $\mathcal{R}.e^S$ that ends in $]_k$, and a regular expression in $\mathcal{R}.e^{]_k}_{[_i^t}$ that ends in $[_i^t$.

iii. there exists $]_k[_k \in N^{(3)}$ such that there exist a regular expression in $\mathcal{R}.e^S$ that ends in $]_k$, and $]_{k_1}[_{k_1}, \dots,]_{k_m}[_{k_m} \in N^{(3)}$ such that there is a regular expression in $\mathcal{R}.e^{]_k}$ that ends in $]_{k_1}$, a regular expression in $\mathcal{R}.e^{]_{k_1}}$ that ends in $]_{k_2}$, and so on, until a regular expression in $\mathcal{R}.e^{]_{k_{m-1}}}$ ending in $]_{k_m}$ and a regular expression in $\mathcal{R}.e^{]_{k_m}}_{[_i^t}$ ending in $[_i^t$ are reached.

11. - A vertex labeled by $]_i$, $]_i, [_i \in N_r^{(2)}$, is a final vertex in \mathcal{G}_e if either *i.*, *ii.*, or *iii.* holds:

i. there exists a regular expression in $\mathcal{R}.e^S$ that ends in $]_i$,

ii. there exists $[k,]_k \in N^{(3)}$, such that there exist a regular expression in $\mathcal{R}.e^S$ that ends in $]_k$, and a regular expression in $\mathcal{R}.e^{]_k}$ that ends in $]_i$.

iii. there exists $[k,]_k \in N^{(3)}$ such that there exist a regular expression in $\mathcal{R}.e^S$ that ends in $]_k$, and $[k_1,]_{k_1}, \dots, [k_m,]_{k_m} \in N^{(3)}$ such that there exist a regular expression in $\mathcal{R}.e^{]_k}$ ending in $]_{k_1}$, a regular expression in $\mathcal{R}.e^{]_{k_1}}$ ending in $]_{k_2}$, and so on, until a regular expression in $\mathcal{R}.e^{]_{k_{m-1}}}$ ending in $]_{k_m}$ and a regular expression in $\mathcal{R}.e^{]_{k_m}}$ ending in $]_i$ are reached.

Denote by \mathcal{R}_e the set of all regular expressions obtained by reading all paths in \mathcal{G}_e from the initial vertex S to all final vertices (i.e., all terminal paths). We have

Theorem 3.5. (*Chomsky-Schützenberger theorem*) For each context-free language L there exist an integer K , a regular set R , and a homomorphism h , such that $L = h(D_K \cap R)$. Furthermore, if G is the context-free grammar that generates L , G_k the Dyck normal form of G , and G_k has no extended grammar, then $K = k$ and $D_K \cap R = \mathbb{L}(G_k)$. Otherwise, there exists $p > 0$ such that $K = k + p$, and $D_K \cap R = D'_K$, where D'_K is the subset of D_K computed as in Theorem 2.9.

Proof. Let $G_k = (N_k, T, P_k, S)$ be the Dyck normal form of G such that $L = L(G)$. Suppose that G_k does not have an extended grammar. Let $h_k: \tilde{N}_k \cup \{[i,]_i \mid [i,]_i \in N_r^{(2)} \cup N^{(3)}\} \cup \{S\} \rightarrow \{[i,]_i \mid [i,]_i \in N_r^{(2)} \cup N^{(3)}\} \cup \{[i,]_i \mid [i,]_i \in N_l^{(2)} \cup N^{(1)}\} \cup \{\lambda\}$ be the homomorphism defined by $h_k(S) = \lambda$, $h_k([i,]_i) = [i,]_i$ for $[i,]_i \in N_r^{(2)} \cup N^{(3)}$, $h_k([i,]_i) = [i,]_i$ for $[i,]_i \in N_l^{(2)}$, and $h_k([i,]_i) = [i,]_i$ for $[i,]_i \in N^{(1)}$. Then $R = h_k(\mathcal{R}_e)$ is a regular language such that $D_k \cap R = \mathbb{L}(G_k)$.

To prove the last equality, notice that each terminal path in a dependency graph \mathcal{G}^X (Construction 3.3) provides a string equal to a substring (or a prefix if $X = S$) of a trace-word in $\mathbb{L}(G_k)$ (in which left brackets in $N_l^{(2)}$ are omitted) generated (in the leftmost derivation order) from the derivation time when X is rewritten, up to the moment when the very first left bracket of a pair in $N^{(1)}$ is rewritten. This string corresponds to a regular expression $r.e_{[i,]_i}^{(l, X)} \in \mathcal{R}_{[i,]_i}^X$, which is extended with another regular expression $r.e_{[i,]_i}^{(r, X)}$ that is the “mirror image” of left brackets in $N_r^{(2)} \cup N^{(3)}$ occurring in $r.e_{[i,]_i}^{(l, X)}$. If left brackets in $N_r^{(2)} \cup N^{(3)}$ are enrolled in a star-height, then their homomorphic image (through h_G^r) in $r.e_{[i,]_i}^{(r, X)}$ is another star-height. The “mirror image” of consecutive left brackets in $N_r^{(2)}$ (with respect to their relative core) is a segment composed of consecutive right brackets in $N_r^{(2)}$. The “mirror image” of consecutive left brackets in $N^{(3)}$ is “broken” by the interpolation of a regular expression $r.e_{[i,]_i}^{]_j}$ in $\mathcal{R}.e^{]_j}$, $[j,]_j \in N^{(3)}$. The number of $r.e_{[i,]_i}^{]_j}$ insertions matches the number of left brackets $[j,]_j$ placed at the left side of the relative core (this is assured by the intersection with D_k). In fact, the extended dependency graph of G_k has been conceived such that it reproduces, on regular expressions in \mathcal{R}_e , the structure of trace-words in $\mathbb{L}(G_k)$. The main problem is the “star-height synchronizations” for brackets in $N_r^{(2)} \cup N^{(3)}$, i.e., the number of left-brackets occurring in a loop placed at the left-side of a core segment $[i,]_i^t$, to be equal to the number of their pairwise right-brackets occurring in the corresponding “mirror” loop placed at the right-side of its relative core, $[i,]_i^t$, $[i,]_i^t \in N^{(1)}$. This is controlled

by the intersection of $h_k(\mathcal{R}_e)$ with D_k , leading to $L(G_k)$. In few words, the proof is by the construction described in Construction 3.4. Another problem that occurs is that the construction of \mathcal{G}_e allows to concatenate $r.e_{[i]}^{(l,X)} \in \mathcal{R}_{[i]}^X$ to its right pairwise $r.e_{[i]}^{(r,X)}$ as well as to another regular expression $\overline{r.e}_{[i]}^{(r,X')}$ (which by construction it is also concatenated to its left pairwise $\overline{r.e}_{[i]}^{(l,X')}$) where X and X' are not necessarily distinct. This does not change the intersection with the Dyck language, but enlarges the regular language $R = h_k(\mathcal{R}_e)$ with useless⁹ words.

If G_k has an extended grammar $G_{k+p} = (N_{k+p}, T, P_{k+p}, S)$, built as in the proof of Theorem 2.9, then \mathcal{R}_e is augmented with $\nabla_e = \{S[t_{k+1}, \dots, S[t_{k+p}]\}$ and h_k is extended to $h_K : \tilde{N}_k \cup \{S\} \cup \{[i]_i | [i]_i \in N_r^{(2)} \cup N^{(3)}\} \cup \{[t_{k+1}, \dots, t_{k+p}] \rightarrow \{[i]_i | [i]_i \in N_r^{(2)} \cup N^{(3)}\} \cup \{[i]_i | [i]_i \in N_l^{(2)} \cup N^{(1)}\} \cup \{[t_{k+1}]_{t_{k+1}}, \dots, [t_{k+p}]_{t_{k+p}}\} \cup \{\lambda\}$, where $h_K(x) = h_k(x)$, $x \notin \{[t_{k+1}, \dots, t_{k+p}]\}$, $h_K([t_{k+j}]) = [t_{k+j}]_{t_{k+j}}$, $1 \leq j \leq p$, $K = k + p$. $L(G_k)$ is augmented with $L_p = \{[t_{k+1}]_{t_{k+1}}, \dots, [t_{k+p}]_{t_{k+p}}\}$ and $D'_K = h_K(\mathcal{R}_e \cup \nabla_e) \cap D_K = L(G_k) \cup L_p$.

The homomorphism h is equal to φ in Theorem 2.9, i.e., $\varphi : (N_{k+p} - \{S\})^* \rightarrow T^*$, $\varphi(N) = \lambda$, for each rule of the form $N \rightarrow XY$, $N, X, Y \in N_k$, and $\varphi(N) = t$, for each rule of the form $N \rightarrow t$, $N \in N_k - \{S\}$, $t \in T$, $\varphi([_{k+i}) = t_{k+i}$, and $\varphi(]_{k+i}) = \lambda$, for each $1 \leq i \leq p$. \square

Note that, for the case of linear languages there is only one dependency graph \mathcal{G}^S . The regular language in the Chomsky-Schützenberger theorem can be built without the use of the extended dependency graph. It suffices to consider only the regular expressions in $\mathcal{R}.e^S = \bigcup_{[i]_i, [j]_j \in N^{(1)}} \mathcal{R}.e_{[i]}^S$. If G_k has an extended grammar G_K , then $L(G_k) = \varphi(D_K \cap h_K(\mathcal{R}.e^S \cup \nabla_e))$, where $K = k + p$, G_K , ∇_e , and φ are defined as in Theorems 2.9 and 3.5. If G_k has no extended grammar then $L(G_k) = \varphi(D_k \cap h_k(\mathcal{R}.e^S))$. However, a graphical representation may be considered an interesting common framework for both, linear and context-free languages. Below we illustrate the manner in which the regular language in the Chomsky-Schützenberger theorem can be computed for linear (Examples 3.6) and context-free (Example 3.7) languages.

Example 3.6. Consider the linear context-free grammar $G = (\{S, [1 \dots, [7,]_1 \dots,]_7\}, \{a, b, c, d\}, S, P)$ in linear-Dyck normal form, with $P = \{S \rightarrow [1]_1,]_1 \rightarrow [2]_2^t, [2]_2 \rightarrow [3]_3^t, [3]_3 \rightarrow [2]_2^t/[4]_4,]_4 \rightarrow [5]_5^t, [5]_5 \rightarrow [6]_6^t, [6]_6 \rightarrow [1]_1/[7]_7^t, [1]_1 \rightarrow a,]_2^t \rightarrow b, [3]_3 \rightarrow c, [4]_4 \rightarrow b, [5]_5 \rightarrow d, [6]_6 \rightarrow b, [7]_7 \rightarrow a,]_7^t \rightarrow a\}$.

The dependency graph \mathcal{G}^S and extended dependency graph \mathcal{G}_e of G are depicted in Figure 1.a and 1.b, respectively. There exists only one regular expression readable from \mathcal{G}^S , i.e., $r.e_{[7]}^{(l,S)} = S([1]([2[3]^+]_4[5][6]^+]_7)^+]$. Hence, $r.e_{[7]}^S = r.e_{[7]}^{(l,S)} r.e_{[7]}^{(r,S)} = S([1]([2[3]^+]_4[5][6]^+]_7)^+]$.

The regular language provided by the Chomsky-Schützenberger theorem is

$$R = ([1]_1([2]_2[3]_3^+ [4]_4 [5]_5 [6]_6)^+ [7]_7^t ([5]_5 ([3]_3 [2]_2)^+)^+)^+.$$

Therefore, $D'_7 = D_7 \cap R = \{([1]_1([2]_2[3]_3^n [4]_4 [5]_5 [6]_6)^m [7]_7^t ([5]_5 ([3]_3 [2]_2)^n)^m | n, m \geq 1\} = L(G_k)$, and $L(G) = \varphi(D'_7) = \{(abb)^m aa(dcb)^n | n, m \geq 1\}$ (G contains no rule of the form $S \rightarrow t$, $t \in T$).

⁹In Section 4 we show how these unnecessary concatenations can be avoided, through a refinement procedure of the regular language in the Chomsky-Schützenberger theorem.

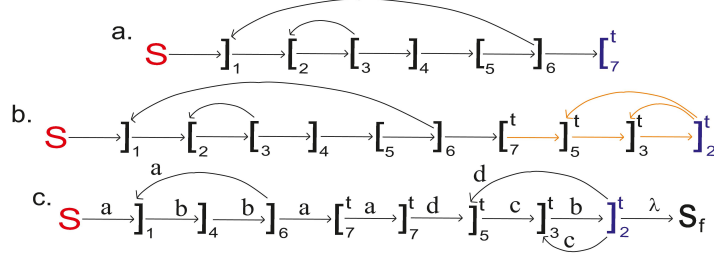


Figure 1: a. The dependency graph \mathcal{G}^S of grammar G in Example 1. b. The extended dependency graph of G . Edges colored in orange extend \mathcal{G} to \mathcal{G}_e . c. The transition diagram \mathcal{A}_e (see Example 5.1 a.) built from \mathcal{G}_e . Each bracket $[i (S,]i)$ in \mathcal{A}_e corresponds to state $s_{[i}$ ($s_S, s_{[i}$). In all graphs S is the initial vertex. In a. - b. the vertex colored in blue is the final vertex.

Example 3.7. Consider the context-free grammar $G = (\{S, [1 \dots, [7,]1 \dots,]7\}, \{a, b, c\}, S, P)$ in Dyck normal form with $P = \{S \rightarrow [1]_1, [1 \rightarrow [5]_5^t/[1]_1,]_1 \rightarrow [6]_6, [2 \rightarrow [6]_6/[7]_7^t, [3 \rightarrow [7]_7^t, [5 \rightarrow [4]_4^t, [6 \rightarrow [3]_3^t,]_6 \rightarrow [2]_2^t,]_7 \rightarrow [3]_3^t/[4]_4^t,]_2^t \rightarrow b,]_3^t \rightarrow a, [4 \rightarrow c, [4 \rightarrow c,]_4^t \rightarrow c,]_5^t \rightarrow b, [7 \rightarrow a\}$

The sets of regular expressions and extended regular expressions obtained by reading \mathcal{G}^S (Figure 2.a) are $\mathcal{R}_{[4}^S = \{S[1^+[5[4]^t\}$ and $\mathcal{R}.e^S = \mathcal{R}.e_{[4}^S = \{S[1^+[5[4]^t]_1^+\}$, respectively.

The regular expressions and extended regular expressions readable from $\mathcal{G}^{]1}$ (Figure 2.b) are $\mathcal{R}_{[4}^{]1} = \{[1[6([3]_7)^+[4]^t\}$ and $\mathcal{R}.e^{]1} = \{[1[6([3]_7)^+[4]^t([3]_3^t)^+]_6\}$, respectively. The regular expressions and extended regular expressions obtained by reading $\mathcal{G}^{]6}$ (Figure 2.c) are $\mathcal{R}_{[4}^{]6} = \{[6[2[6([3]_7)^+[4]^t([3]_3^t)^+]_6]_2^t, [6[2([7]_3)^*]_7[4]^t([3]_3^t)^*]_2^t\}$, respectively.

The extended dependency graph of G is sketched in Figure 2.d. Edges in black, are built from the regular expressions in $\mathcal{R}_{[4}^X$, $X \in \{S,]1,]6\}$. Orange edges emphasize symmetrical structures, built with respect to the structure of trace-words in $L(G)$. Some of them (e.g., $]_2^t]_1$ and $]_2^t]_2^t$) connect regular expressions in \mathcal{R}_e between them with respect to the structure of trace-words in $L(G)$ (see Construction 3.4, item 8). The edge $]_2^t]_1$ is added because there exists at least one regular expression in \mathcal{R}_e that contains $]_1]_1$ (e.g. $S[1^+[5[4]^t]_1^+$), a regular expression in $\mathcal{R}.e_{[4}^{]1}$ that ends in $]_6$ (e.g. $]_1[6([3]_7)^+[4]^t([3]_3^t)^+]_6$) and a regular expression in $\mathcal{R}.e_{[4}^{]6}$ that ends in $]_2^t$ (see Construction 3.4, item 8.iii.). The $+$ self-loop $]_2^t]_2^t$ is due to the existence of a regular expression that contains $]_6]_2^t$ (e.g. $]_6[2[6([3]_7)^+[4]^t([3]_3^t)^+]_6]_2^t$) and a regular expression in $\mathcal{R}.e_{[4}^{]6}$ that ends in $]_2^t$ (e.g. $]_6[2[6([3]_7)^+[4]^t([3]_3^t)^+]_6]_2^t$ or $]_6[2([7]_3)^*]_7[4]^t([3]_3^t)^*]_2^t$).

The regular language provided by the Chomsky-Schützenberger theorem is the homomorphic image, through h_k (defined in Theorem 3.5), of all regular expressions associated with all paths in the extended dependency graph in Figure 2.d, reachable from the initial vertex S to the final vertex labeled by $]_2^t$, i.e., terminal paths.

The interpretation that emerges from the graphical method described in this paper is that *the regular language in the Chomsky-Schützenberger theorem intersected with a (certain) Dyck language lists all derivation trees (read in the depth-first search order) associated with words in a context-free grammar, in Dyck normal form or in Chomsky normal form*

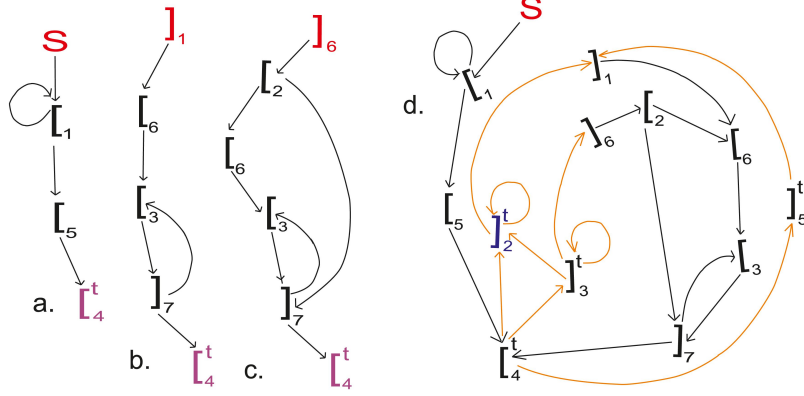


Figure 2: a. - d. The dependency graphs of the context-free grammar G in Example 3.7. e. The extended dependency graph of G . In all graphs, vertices colored in red are initial vertices, while vertices colored in blue are final vertices. Edges colored in orange, in d , emphasize symmetrical structures obtained by linking the dependency graphs between them.

(since these derivation trees are equal, up to an homomorphism). The intersection forms (with very little exceptions) the trace-language associated with the respective context-free grammar.

In the next section we refine the extended dependency graph \mathcal{G}_e to provide a thinner regular language in the Chomsky-Schützenberger theorem with respect to the structure of the context-free grammar in Dyck normal form obtained through the algorithm described in the proof of Theorem 1.2. Based on this readjustment in Section 5 we sketch a *transition diagram* for a *finite automaton* and a *regular grammar* that generates a *regular superset approximation* for the initial context-free language.

4 Further Refinements of the Regular Language in the Chomsky-Schützenberger Theorem

One of the main disadvantage of considering $*$ -height regular expressions in building the extended dependency graph associated with a context-free grammar in Dyck normal form is that some $*$ -loops composed of right brackets in $N_r^{(2)} \cup N^{(3)}$ may not be symmetrically arranged according to their corresponding left brackets in $N_r^{(2)} \cup N^{(3)}$, if we consider their corresponding core segment as a symmetrical center. This is due to the possibility of having “ λ -loops”. This deficiency does not affect the intersection with a Dyck language, but it has the disadvantage of enlarging considerable the regular language in the Chomsky-Schützenberger theorem. This can be avoided by considering only loops described in terms of $+$ Kleene closure.

Another disfunction of the extended dependency graph built through Construction 3.4 is the concatenation of a regular expression $r.e_{[i]}^{(l,X)}$ with another regular expression $\overline{r}.e_{[i]}^{(r,X')}$,

$\overline{r.e}_{[i]_i^t}^{(r,X')} \neq r.e_{[i]_i^t}^{(r,X)}$ (due to the common tie $[i]_i^t$ that marks a core segment). This can be avoided by a renaming procedure of the regular expressions we want to concatenate. All these additional modifications in building an extended dependency graph are useful only if we want to refine the regular language that satisfies the Chomsky-Schützenberger theorem (with regards to the grammar in Dyck normal form). This will be further handled (in Section 5) to build a tighter approximation for the context-free language it characterizes.

Each regular expression of a finite star-height can be described as a finite union of regular expressions in terms of $+$ Kleene closure (shortly plus-height). For instance the $*$ -height regular expression $]_6[2(]_7[3)^*]_7[_4^t$ in $\mathcal{R}_{[i]_i^t}^6$ can be forked into $]_6[2(]_7[3)^+]_7[_4^t$ and $]_6[2]_7[_4^t$. The plus-height of a regular expression, can be defined analogous to the star-height of a regular expression in [18], as follows.

Definition 4.1. Let Σ be a finite alphabet. The plus-height $h(r)$ of a regular expression r is defined recursively as follows: *i.* $h(\lambda) = h(\emptyset) = h(a) = 0$ for $a \in \Sigma$, *ii.* $h(r_1 \cup r_2) = h(r_1 r_2) = \max\{h(r_1), h(r_2)\}$, and $h(r^+) = h(r) + 1$.

Note that for any star-height regular expression it is possible to build a digraph, with an initial vertex v_i and a final vertex v_f , such that all paths in this digraph, from v_i to v_f , to provide the respective regular expression (which can be done in a similar manner as in Construction 3.4). However, if the regular expression is described in terms of plus-height then this statement may not be true (due to the repetition of some symbols). To force this statement be true, also for plus-height regular expressions, each repetition of a bracket is marked by a distinct symbol (e.g., $]_6[2(]_7[3)^+]_7[_4^t$ becomes $]_6[2(]_7[3)^+]_7[_4^t$), and then, for the new plus-height regular expression obtained in this way, we build a digraph with the above property. In order to recover the initial plus-height regular expression from the associated digraph, a homomorphism that maps all the marked brackets (by distinct symbols) into the initial one must be applied. Each time it is required, we refer to such a vertex as a h -marked vertex. Therefore, due to the technical transformations described above and the symmetrical considerations used in the construction of a trace language, we may assume to work only with plus-height regular expressions.

Let $G_k = (N_k, T, P_k, S)$ be an arbitrary context-free grammar in Dyck normal form, and \mathcal{G}^X the dependency graph of G_k (see Construction 3.3). Denote by $\mathcal{P}_{[i]_i^t}^X$ the set of all plus-height regular expressions over $\tilde{N}_k \cup \{X\}$ that can be read in \mathcal{G}^X , starting from the initial vertex X and ending in the final vertex $[i]_i^t$. The cardinality of $\mathcal{P}_{[i]_i^t}^X$ is finite. Now, we consider the same homomorphism, as defined for the case of the set $\mathcal{R}_{[i]_i^t}^X$, i.e., $h_{\mathcal{G}}: \tilde{N}_k \cup \{X\} \rightarrow \{[i]_i,]_i \in N_r^{(2)} \cup N^{(3)}\} \cup \{\lambda\}$ such that $h_{\mathcal{G}}([i]_i) =]_i$ for any pair $[i]_i \in N_r^{(2)} \cup N^{(3)}$, $h_{\mathcal{G}}(X) = h_{\mathcal{G}}([i]_i) = h_{\mathcal{G}}(]_i) = \lambda$, for any $[i]_i,]_i \in N_l^{(2)}$. For any element $r.e_{[i]_i^t}^{(l,X)} \in \mathcal{P}_{[i]_i^t}^X$ we build a new plus-height regular expression $r.e_{[i]_i^t}^{(r,X)} = h_{\mathcal{G}}^r(r.e_{[i]_i^t}^{(l,X)})$, where $h_{\mathcal{G}}^r$ is the mirror image of $h_{\mathcal{G}}$. Consider $r.e_{[i]_i^t}^X = r.e_{[i]_i^t}^{(l,X)} r.e_{[i]_i^t}^{(r,X)}$. For a certain X and $[i]_i^t$, denote by $\mathcal{P}.e_{[i]_i^t}^X$ the set of all (plus-height) regular expressions $r.e_{[i]_i^t}^X$ obtained as above. Furthermore, $\mathcal{P}.e^X = \bigcup_{[i]_i^t \in N^{(1)}} \mathcal{P}.e_{[i]_i^t}^X$, and $\mathcal{P}.e = \mathcal{P}.e^S \cup (\bigcup_{[i]_i \in N^{(3)}} \mathcal{P}.e^i)$.

Note that linear languages do not need an extended dependency graph. The set of all regular expressions $\mathcal{P}.e^S$ suffices to build a regular language in the Chomsky-Schützenberger theorem (see Theorem 3.5) that cannot be further adjusted by using the graphical method proposed in this section. Furthermore $|\mathcal{R}.e^S| \leq |\mathcal{P}.e^S|$. Equality takes place only for the case when each regular expression in $\mathcal{R}.e^S$ is a plus-height regular expression (see Example 3.6). For the case of context-free languages the plus-height regular expressions in $\mathcal{P}.e$ must be linked with each other in such a way it approximates, as much as possible, the trace-language associated with the respective context-free language.

In order to find an optimal connection of the regular expressions in $\mathcal{P}.e$ we consider the following labeling procedure of elements in $\mathcal{P}.e$. Denote by c_0 the cardinality of $\mathcal{P}.e^S$, i.e., $|\mathcal{P}.e^S| = c_0$, and by c_j the cardinality of $\mathcal{P}.e^{[j]}$, where $[j,]_j \in N^{(3)}$. Each regular expression $r \in \mathcal{P}.e^S$ is labeled by a unique q , $1 \leq q \leq c_0$, and each regular expression $r \in \mathcal{P}.e^{[j]}$, is labeled by a unique q , such that $\sum_{r=0}^{i-1} c_r + 1 \leq q \leq \sum_{r=0}^i c_r$, $1 \leq i \leq s$, and $s = |\{[j,]_j | j \in N^{(3)}\}|$. Denote by r^q the labeled version of r . To preserve symmetric structures that characterize trace-words of context-free languages, then when we link regular expressions in $\mathcal{P}.e$ between them, each bracket in a regular expression r^q is upper labeled by q . Exception makes the first bracket occurring in r^q (which is a bracket in $\{[j,]_j | j \in N^{(3)}\}$). Now, a refined extended digraph can be built similar to that described in Construction 3.4.

To have a better picture of how the labeled regular expressions must be linked to each other, and where further relabeling procedures may be required (to obtain a better approximation of the trace-language), we first build for each plus-height regular expression $r^q \in \mathcal{P}.e^{[j]}$, $[j,]_j \in N^{(3)}$, a digraph and then we connect all digraphs between them. Denote by $G^{q,[j]}$ the digraph associated with $r^q \in \mathcal{P}.e^{[j]}$, such that $]_j$ is the initial vertex and the final vertex is the last bracket occurring in r^q . Each digraph $G^{q,[j]}$ read from the initial vertex $]_j$ to the final vertex provides the regular expression r^q . Hence, any digraph $G^{q,[j]}$ has vertices labeled by brackets of the forms $\{[j^q,]_j | j \in N^{(1)} \cup N_r^{(2)} \cup N^{(3)}\} \cup \{[j^q,]_j | j \in N_l^{(2)} \cup N_r^{(2)} \cup N^{(3)}\}$, $c_0 \leq q \leq \sum_{r=0}^s c_r$, with the exception of the initial vertex $]_j$, $[j,]_j \in N^{(3)}$. Some of vertices in $G^{q,[j]}$, besides the q -index, may also be \hbar -marked, in order to prevent repetitions of the same vertex which may occur in a plus-height regular expression. As the construction of the dependency graph does not depend on \hbar -markers, unless it is necessary, we avoid \hbar -marked notations in further explanations when building this digraph.

The adjacent vertex Y to $]_j$, in $G^{q,[j]}$, is called *sibling*. Any edge of the form $]_l^-]_k^-$, where $[l,]_l \in N^{(3)}$, $[k,]_k \in N_r^{(2)} \cup N^{(3)}$, is called *dummy edge*, while $]_l^- (]_k^-$, if $[k,]_k \in N^{(3)}$ is a *dummy vertex*. An edge that is not a dummy edge is called *stable edge*. Denote by $\mathcal{G}^{[j]}$ the set of all digraphs $G^{q,[j]}$, i.e., their initial vertex is $]_j$. Any digraph $G^{q,[j]}$ has only one bracket $[k^q,]_k \in N^{(1)}$, which stands for a core segment in a trace-word. Right brackets $]_j^q$, $[j,]_j \in N_r^{(2)} \cup N^{(3)}$, must be symmetrically arranged according to their left pairwise $]_j^q$, $[j,]_j \in N_r^{(2)} \cup N^{(3)}$, that occur at the left side of $[k^q,]_k$. A dummy vertex labeled by $]_j^q$, $[j,]_j \in N^{(3)}$, allows the connection with any digraph in $\mathcal{G}^{[j]}$. A digraph in $\mathcal{G}^{[j]}$ with a final vertex labeled by a bracket $[k^-,]_k \in N^{(1)}$, or by a bracket $]_l^-$, $[l,]_l \in N_r^{(2)}$, is called *terminal*, because the vertex $[k^-]$ or $]_l^-$, respectively, does not allow more connections.

Next we describe the procedure that builds a refined extended digraph with the property

that reading this digraph (in which each loop is a plus-loop) from the initial vertex (which is S) to all its final vertices, we obtain those (plus-height) regular expressions that form a regular language that provides the best approximation of the corresponding trace-language.

Step 1. First we build a digraph $\mathcal{G}.e^S$ that describes all (plus-height) regular expressions in $\mathcal{P}.e^S$. This can be done by connecting all digraphs in \mathcal{G}^S to S . Since each bracket labeling a vertex in $G^{q,S}$, $1 \leq q \leq c_0$, is uniquely labeled by q , and there exists a finite number of brackets, $\mathcal{G}.e^S$ is correct (in the sense that it is finite and any vertex occurs only one time). The initial vertex of $\mathcal{G}.e^S$ is S . If a graph in \mathcal{G}^S has a final vertex labeled by a bracket $]_i^q$, $[i,]_i \in N^{(1)}$ or by a bracket $]_i^q$, $[i,]_i \in N_r^{(2)}$, then this is also a final vertex in $\mathcal{G}.e^S$.

If G_k is a grammar in linear-Dyck normal form then $\mathcal{G}.e^S$, built in this way, suffices to build the regular language in the Chomsky-Schützenberger theorem. The set of all paths from S to each final vertex to which we apply the homomorphism h_k , defined in the proof of Theorem 3.5, yields a regular language R_m that cannot be further adjusted, such that the Chomsky-Schützenberger theorem still holds. Therefore, we call the R_m language, as minimal with respect to the grammar G_k and the Chomsky-Schützenberger theorem, i.e., the equality $\varphi(D_K \cap R_m) = \varphi(L(G_k))$ still holds, where φ is the homomorphism defined in the proof of Theorem 2.9.

Step 2. For each vertex $]_j^q$ existing in $\mathcal{G}.e^S$, such that $]_j \in N^{(3)}$, we connect all digraphs in $\mathcal{G}^{]_j}$ to $\mathcal{G}.e^S$. This can be done by adding to $\mathcal{G}.e^S$ a new edge $]_j^q Y$, for each sibling Y of $]_j$ (in $\mathcal{G}^{]_j}$). If Z is the adjacent vertex of $]_j^q$ (in the former version of $\mathcal{G}.e^S$), i.e., $]_j^q Z$ is a dummy edge, then we remove in $\mathcal{G}.e^S$ the edge $]_j^q Z$, while in $G^{q',]_j}$ (connected to $\mathcal{G}.e^S$ through $]_j^q$) we remove the vertex $]_j$ and consequently, the edge $]_j Y$. For the moment, all the other edges in $G^{q',]_j}$ are preserved in $\mathcal{G}.e^S$, too. Besides, if V is the final vertex of $G^{q',]_j}$, then a new edge VZ is added to $\mathcal{G}.e^S$. If $V \in \{[k^- | [k,]_k \in N^{(1)}\} \cup \{[l^- | [l,]_l \in N_r^{(2)}\}$, i.e., $G^{q',]_j}$ is a terminal digraph then the edge VZ is a *glue edge*, i.e., it is a stable edge that makes the connection of $G^{q',]_j}$ into $\mathcal{G}.e^S$ (or more precisely the connection of $G^{q',]_j}$ to $G^{q',]_j}$ digraph in which it has been inserted). Otherwise, VZ is a dummy edge, which will be removed at a further connection with a digraph in \mathcal{G}^V . Since for the case of linear languages generated by a grammar in linear-Dyck normal form, $\mathcal{G}.e^S$ does not contain any dummy vertex, the construction of $\mathcal{G}.e^S$ is completed at *Step 1*.

A vertex in $\mathcal{G}.e^S$ labeled by a bracket $]_j^q$, $[j,]_j \in N^{(3)}$, that has no adjacent vertex, i.e., the out degree of the vertex labeled by $]_j^q$ is 0, is called *pop vertex*. When connecting a digraph $G^{q',]_j}$ to $\mathcal{G}.e^S$, through a pop vertex, if $G^{q',]_j}$ is a terminal digraph, then the final vertex of $G^{q',]_j}$ becomes a final vertex of $\mathcal{G}.e^S$. If $G^{q',]_j}$ is not a terminal digraph, then the final vertex of $G^{q',]_j}$ becomes a pop vertex for $\mathcal{G}.e^S$.

If there exist more than one vertex labeled by an upper indexed¹⁰ bracket $]_j^{\bar{q}}$, $[j,]_j \in N^{(3)}$, then, if $G^{q',]_j}$ has been already added to $\mathcal{G}.e^S$ there is no need to add another “copy” of $G^{q',]_j}$. It is enough to connect $]_j^{\bar{q}}$ to the digraph existing in $\mathcal{G}.e^S$, i.e., to add a new edge $]_j^{\bar{q}} Y$, where Y is a sibling of $]_j$ in $G^{q',]_j}$. This observation holds for any element in $\mathcal{G}^{]_j}$. The procedure described at *Step 2* is repeated for each new dummy or pop vertex added

¹⁰As $\mathcal{G}.e^S$ is finite, there cannot exist in $\mathcal{G}.e^S$ two right brackets $]_j$, $[j,]_j \in N^{(3)}$, upper indexed by the same value.

to $\mathcal{G}.e^S$. For each transformation performed on $\mathcal{G}.e^S$, we maintain the same notation $\mathcal{G}.e^S$ for the new obtained digraph. The construction of $\mathcal{G}.e^S$ ends up then when each vertex $]_j^-, [j,]_j \in N^{(3)}$, has been connected to a digraph in \mathcal{G}^{lj} , i.e., no dummy and pop vertices exist in $\mathcal{G}.e^S$. The only permissible contexts under which a bracket $]_j^-, [j,]_j \in N^{(3)}$, may occur, in the final version of $\mathcal{G}.e^S$, are of the forms $]_i^-]_j^- [k, [h]_j^- [k,]_i^-]_j^-]_l^-$, $[h]_j^-]_l^-$, where $[i,]_i \in N_r^{(2)}$, $[h,]_h \in N^{(1)}$, $[k,]_k \in N^{(1)} \cup N_r^{(2)} \cup N^{(3)}$, $[l,]_l \in N_l^{(2)}$.

There are several refinements that can be done on $\mathcal{G}.e^S$ such that the resulted regular language better approximates the trace language associated with the considered context-free language. Two peculiar situations may occur when adding digraphs to $\mathcal{G}.e^S$:

\mathcal{I}_1 . First, suppose that during the construction of $\mathcal{G}.e^S$ by subsequently connecting digraphs between them, starting from $]_j^q, [j,]_j \in N^{(3)}$, we reach a terminal digraph with a final vertex $]_k^{q'}, [k,]_k \in N_r^{(2)}$, such that $]_k^{q'}$ is linked to Z , forming thus a stable (glue) edge $]_k^{q'} Z$. Denote by $\wp =]_j^q \dots]_k^{q'} Z$ the path (in $\mathcal{G}.e^S$) from $]_j^q$ to $]_k^{q'} Z$, obtained at this stage. If the vertex that precedes $]_k^{q'}$ in \wp is $]_j^{q'}, [j,]_j \in N^{(3)}$, i.e., $\wp =]_j^q \dots]_j^{q'}]_k^{q'} Z$, then connecting $]_j^{q'}$ ($]_j^{q'}]_k^{q'}$ is a dummy edge), through its siblings, to digraphs in \mathcal{G}^{lj} another edge $]_k^{q'}]_k^{q'}$ preceded by $]_j^{q'}]_k^{q'}$, is added to $\mathcal{G}.e^S$, i.e., \wp becomes $\wp =]_j^q \dots]_j^{q'} (]_k^{q'})^2 Z$. Since $]_j^{q'}]_k^{q'}$ is a dummy edge, the vertex $]_j^{q'}$ must be again connected to digraphs in \mathcal{G}^{lj} , and so on, until $]_j^{q'}$ is connected to a terminal digraph $G^{\bar{q},lj} \in \mathcal{G}^{lj}$, $\bar{q} \neq q'$, that has a final vertex labeled by a bracket $]_{\bar{m}}^{\bar{q}}, [m,]_{\bar{m}} \in N_r^{(2)}$ (m and k not necessarily distinct), or by a bracket $]_{\bar{m}}^{\bar{q}}, [m,]_{\bar{m}} \in N^{(1)}$ such that $]_{\bar{m}}^{\bar{q}}$ is not preceded by a bracket of the form $]_j^-, [j,]_j \in N^{(3)}$. Then \wp will be either of the form $]_j^q \wp_1]_{\bar{m}}^{\bar{q}} (]_{\bar{m}}^{\bar{q}})^+ Z$ or of the form $]_j^q \wp_1]_{\bar{m}}^{\bar{q}} (]_{\bar{m}}^{\bar{q}})^+ Z$, respectively. On the other hand, since $G^{\bar{q},lj} \in \mathcal{G}^{lj}$ the digraph $G^{\bar{q},lj}$ can be added to $\mathcal{G}.e^S$, through $]_j^q$, from the very first beginning, avoiding thus the plus-loop $(]_{\bar{m}}^{\bar{q}})^+$, i.e., there should exist in $\mathcal{G}.e^S$ a new path $\wp' =]_j^q \wp_2]_{\bar{m}}^{\bar{q}} Z$ or $\wp' =]_j^q \wp_2]_{\bar{m}}^{\bar{q}} Z$ (where \wp_2 is a path in $G^{\bar{q},lj}$). This allows two other new paths to be created in $\mathcal{G}.e^S$, i.e., $\bar{\wp} =]_j^q \wp_2]_{\bar{m}}^{\bar{q}} (]_{\bar{m}}^{\bar{q}})^+ Z$ (or $\wp'' =]_j^q \wp_2]_{\bar{m}}^{\bar{q}} (]_{\bar{m}}^{\bar{q}})^+ Z$) and $\bar{\wp}' =]_j^q \wp_1]_{\bar{m}}^{\bar{q}} Z$ (or $\bar{\wp}'' =]_j^q \wp_1]_{\bar{m}}^{\bar{q}} Z$), which are of no use in approximating the trace language (hence in building the regular language in the Chomsky-Schützenberger theorem). Paths $\bar{\wp}$ and $\bar{\wp}'$ (\wp'' , $\bar{\wp}''$) do not affect the intersection with the Dyck language but they enlarge the regular language with useless words.

In order to avoid the paths $\bar{\wp}$ and $\bar{\wp}'$ (or \wp'' , $\bar{\wp}''$) the terminal digraph $G^{\bar{q},lj}$ receives a new label \tilde{q} , besides of label \bar{q} (which is maintained to allow \wp to be produced). To allow the shorter path \wp' to be created, instead of $G^{\bar{q},lj}$ the terminal digraph $G^{\tilde{q},lj}$ is connected to $\mathcal{G}.e^S$ through the dummy vertex $]_j^q$. Hence \wp' becomes $\wp' =]_j^q \dots]_{\bar{m}}^{\tilde{q}} Z$ (or $\wp' =]_j^q \dots]_{\bar{m}}^{\tilde{q}} Z$), while \wp remains $]_j^q \dots]_{\bar{m}}^{\tilde{q}} (]_{\bar{m}}^{\tilde{q}})^+ Z$ (or $]_j^q \dots]_{\bar{m}}^{\tilde{q}} (]_{\bar{m}}^{\tilde{q}})^+ Z$, respectively). This relabeling procedure is used for any case similar to that described above¹¹ encountered during the computation of $\mathcal{G}.e^S$. As there may exist a finite number¹² of plus-loops in $\mathcal{G}.e^S$, there will be a finite number of

¹¹For instance, $]_k^{q'}$ may also be a dummy vertex and $]_k^{q'} Z$ a dummy edge.

¹²The plus-height of a regular expression obtained from any digraph in \mathcal{G}^{lj} is finite related to the length of the strings in $L(G_k)$.

“reabeled” digraphs (not necessarily terminal). A loop (not necessarily a self-loop) may be reached through different paths that must be “renamed” (if we want to avoid that loop).

\mathcal{I}_2 . Another situation that requires a relabeling procedure may occur when connecting a digraph to $\mathcal{G}.e^S$ through a pop vertex. Suppose that $]_j^q, [j,]_j \in N^{(3)}$, is a pop vertex, and the digraph $G^{q',]_j}$ that must be added to $\mathcal{G}.e^S$ has been already connected through a dummy vertex labeled by $]_j^q$ (i.e., $G^{q',]_j}$ has been already inserted in $\mathcal{G}.e^S$). According to the procedure described at *Step 2* the vertex $]_j^q$ is linked to the sibling of $]_j$ in $G^{q',]_j}$ already existing in $\mathcal{G}.e^S$. Since the connection of $G^{q',]_j}$ to $\mathcal{G}.e^S$ has been done through a dummy vertex, the final vertex in $G^{q',]_j}$ cannot be neither a final vertex in $\mathcal{G}.e^S$ (if $G^{q',]_j}$ is a terminal digraph) nor a pop vertex.

To forbid a pop vertex $]_j^-$ to overlap with a dummy vertex $]_j^-$, each of the digraphs connected to $\mathcal{G}.e^S$ through a pop vertex, is renamed by a new label. Denote by $\bar{\mathcal{G}}^{]_j}$ the labeled version of $\mathcal{G}^{]_j}$. Then connections through pop vertices will be done by using only digraphs in $\bar{\mathcal{G}}^{]_j}$. However, any dummy vertex $]_j^-$, that is not a pop vertex, obtained by connecting digraphs in $\bar{\mathcal{G}}^{]_j}$ to $\mathcal{G}.e^S$ should be connected to the original digraphs in $\mathcal{G}^{]_j}$, unless a relabeling procedure described at \mathcal{I}_1 is required.

Denote by $\bar{N}_k = \{[i^- | [i,]_i \in N^{(1)} \cup N_r^{(2)} \cup N^{(3)}\} \cup \{]_j^- | [j,]_j \in N_l^{(2)} \cup N_r^{(2)} \cup N^{(3)}\}$, the set of vertices composing $\mathcal{G}.e^S$, in which some brackets may be \hbar -marked (by distinct \hbar -markers). To reach the regular language in the Chomsky-Schützenberger theorem we denote by $\mathcal{R}_{\mathcal{G}}$ the set of all regular expressions obtained by reading $\mathcal{G}.e^S$ from the initial vertex S to any final vertex. First, suppose that G_k does not have an extended grammar. We have $K = k$ and $D'_k = L(G_k)$. Consider the homomorphism $h_k: \bar{N}_k \cup \{S\} \rightarrow \{[i,]_i | [i,]_i \in N_r^{(2)} \cup N^{(3)}\} \cup \{[i]_i | [i,]_i \in N_l^{(2)} \cup N^{(1)}\} \cup \{\lambda\}$, defined by $h_k(S) = \lambda$, $h_k([i^-]_-) = [i]$, $h_k([i^-]_-) = [i]$ for any $[i,]_i \in N_r^{(2)}$, $h_k([i^-]_-) = [i]$ for any $[i,]_i \in N_l^{(2)}$, $h_k([i^-]_-) = [i]$ for any $[i,]_i \in N^{(1)}$. Then $R_m = h_k(\mathcal{R}_{\mathcal{G}})$ is a regular language with $D_k \cap R_m = L(G_k)$. Furthermore, R_m is a strength refinement of R , such that the Chomsky-Schützenberger theorem still holds. This is because when building regular expressions in $\mathcal{P}.e$ each $r.e_{[i]}^{(l, X)}$ is linked only to its right pairwise $r.e_{[i]}^{(r, X)}$ (due to plus-height considerations and labeling procedures). In this way all plus-loops in $r.e_{[i]}^{(l, X)}$ are correctly mirrored (through h_k^r) into its correct pairwise $r.e_{[i]}^{(r, X)}$. The case of λ -loops is taken by the relabeling procedure described at \mathcal{I}_1 . This is also applicable each time we want to fork a path in $\mathcal{G}.e^S$ in order to avoid useless loops on that path. The relabeling procedure \mathcal{I}_2 allows to leave $\mathcal{G}.e^S$ without re-loading another useless path. That is why the regular language R_m built this way is a tighter approximation of $L(G_k)$. A finer language than R_m can be found by searching for a more efficient grammar in Dyck normal form, with respect to the number of rules and nonterminals.

If G_k has an extended grammar $G_{k+p} = (N_{k+p}, T, P_{k+p}, S)$ (built as in the proof of Theorem 2.9) then $\mathcal{R}_{\mathcal{G}}$ is augmented with $\nabla_e = \{S[t_{k+1}, \dots, S[t_{k+p}\}$ and h_k is extended to h_K , $h_K: \bar{N}_k \cup \{S\} \cup \{[t_{k+1}, \dots, [t_{k+p}\} \rightarrow \{[i,]_i | [i,]_i \in N_r^{(2)} \cup N^{(3)}\} \cup \{[i]_i | [i,]_i \in N_l^{(2)} \cup N^{(1)}\} \cup \{[t_{k+1}]t_{k+1}, \dots, [t_{k+p}]t_{k+p}\} \cup \{\lambda\}$, $h_K(x) = h_k(x)$, $x \notin \{[t_{k+1}, \dots, [t_{k+p}\}$, and $h_K([t_{k+j}]t_{k+j}) = [t_{k+j}]t_{k+j}$, $1 \leq j \leq p$, $K = k + p$.

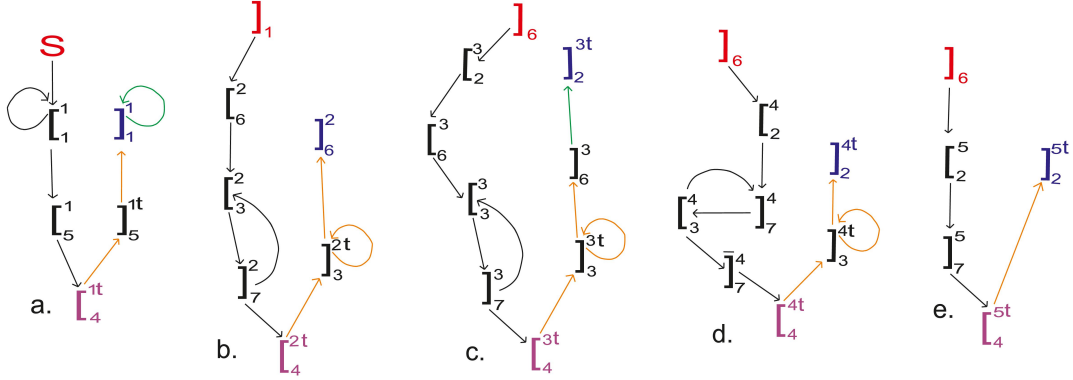


Figure 3: a. - e. Graphs associated with regular expressions in $\mathcal{P.e}$ (Example 4.2). Initial vertices are colored in red, final vertices in blue, while purple vertices mark a core segment. \bar{l}_7^4 is a marked vertex to allow the plus-loop $(\bar{l}_3^4 \bar{l}_7^4)^+$.

Example 4.2. Consider the context-free grammar in Example 3.7 with the dependency graphs sketched in Figure 3. The set $\mathcal{P.e}$ of labeled plus-loop regular expressions built from the dependency graphs is composed of $S(\bar{l}_1^1 + [\bar{l}_5^1 \bar{l}_5^1]_5^1 (\bar{l}_1^1)^+)$ (with the associated digraph $G^{1,S}$, Fig. 3.a), $\bar{l}_1^2 ([\bar{l}_3^2 \bar{l}_7^2]^+ + [\bar{l}_4^2 (\bar{l}_3^2)^+]_6^2)$ (with G^{2,\bar{l}_1} , Fig. 3.b), $\bar{l}_6^3 [\bar{l}_2^3 (\bar{l}_3^3 \bar{l}_7^3)^+ + [\bar{l}_4^3 (\bar{l}_3^3)^+]_6^3 \bar{l}_2^3]$ (with G^{3,\bar{l}_6} , Fig. 3.c), $\bar{l}_6^4 [\bar{l}_2^4 (\bar{l}_7^4 \bar{l}_3^4)^+ + [\bar{l}_4^4 (\bar{l}_3^4)^+]_2^4]$ or the \bar{h} -marked version $\bar{l}_6^4 [\bar{l}_2^4 (\bar{l}_7^4 \bar{l}_3^4)^+ + \bar{l}_7^4 [\bar{l}_4^4 (\bar{l}_3^4)^+]_2^4]$ (with the associated digraph G^{4,\bar{l}_6} , Fig. 3.d), and $\bar{l}_6^5 [\bar{l}_2^5 \bar{l}_7^5 \bar{l}_2^5]$ (with the digraph G^{5,\bar{l}_6} , Fig. 3.e).

The extended dependency graph built with respect to the refinement procedure is sketched in Figure 4. The terminal digraphs G^{6,\bar{l}_6} and G^{7,\bar{l}_6} are introduced with respect to the relabeling procedure \mathcal{I}_1 , in order to prevent the loop yielded by the "iterated" digraph G^{3,\bar{l}_6} to occur between G^{2,\bar{l}_1} and G^{6,\bar{l}_6} (or G^{7,\bar{l}_6}). It also forbids the self-loop $(\bar{l}_2^3)^+$ to be linked to G^{6,\bar{l}_6} (or to G^{7,\bar{l}_6}), then when the digraph G^{3,\bar{l}_6} is not added to the corresponding path. Due to the self-loop $(\bar{l}_1^1)^+$, in which \bar{l}_1^1 is a pop vertex, we did not applied the relabeling procedure described at \mathcal{I}_2 (applying it leads to the same result).

5 A Regular Superset Approximation for Context-Free Languages

A regular language R may be considered a superset approximation for a context-free language L , if $L \subseteq R$. A good approximation for L is that for which the set $R - L$ is as small as possible. There are considerable methods to find a regular approximation for a context-free language. The most significant consist in building, through several transformations applied to the original pushdown automaton (or context-free grammar), the most appropriate finite automaton (regular grammar) recognizing (generating) a regular superset approximation of the original context-free language. How accurate the approximation is, depends on the transformations applied to the considered devices. However, the perfect regular superset (or subset) approximation for an arbitrary context-free language cannot be built. For surveys on approximation methods and their practical applications in computational linguistics (es-

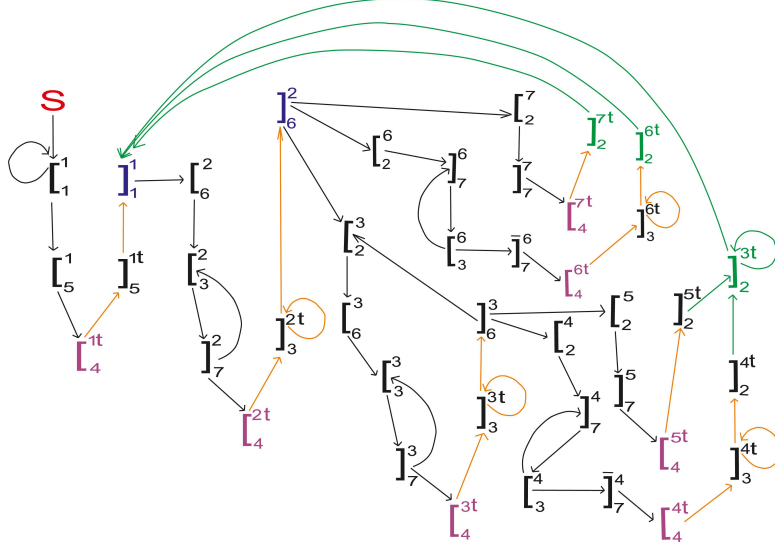


Figure 4: The refined dependency graph of the context-free grammar in Examples 3.7 and 4.2. S is the initial vertex, vertices colored in green are final vertices, vertices colored in blue are dummy vertices, vertices colored in purple mark a core segment. Orange edges emphasize symmetrical structures built with respect to the structure of the trace language. Green edges are glue edges.

pecially in parsing theory) the reader is referred to [21] and [22]. Methods to measure the accuracy of a regular approximation can be found in [4], [8], and [23].

In the sequel we propose a new approximation technique that emerges from the Chomsky-Schützenberger theorem. In brief, the method consists in transforming the original context-free grammar into a context-free grammar in Dyck normal form. For this grammar we build the refined extended dependency graph $\mathcal{G}.e^S$ described in Section 4. From $\mathcal{G}.e^S$ we depict a *state diagram* \mathcal{A}_e for a finite automaton and a regular grammar $G_r = (N_r, T, P_r, S)$ that generates a regular (superset) approximation for $L(G_k)$ (which is nothing else than the image through φ of the language R_m built in Section 4).

Let $G_k = (N_k, T, P_k, S)$ be an arbitrary context-free grammar in Dyck normal form, and $\mathcal{G}.e^S = (V_e, E_e)$ the extended dependency graph of G_k . Recall that $V_e = \{[i^-|[i,]_i \in N^{(1)} \cup N_r^{(2)} \cup N^{(3)}\} \cup \{[j^-|[j,]_j \in N_l^{(2)} \cup N_r^{(2)} \cup N^{(3)}\} \cup \{S\}$ in which some of the vertices may be \hbar -marked, in order to prevent repetition of the same bracket when building the digraph associated with a plus-height regular expression. In brief, the state diagram \mathcal{A}_e can be built by skipping in $\mathcal{G}.e^S$ all left brackets in $N_r^{(2)}$ and all brackets in $N^{(3)}$, and labeling the edges with the symbol produced by left or right bracket in $N^{(2)} \cup N^{(1)}$. This reasoning is applied no matter whether the vertex in V_e is \hbar -marked or not. Therefore, we avoid \hbar -marker specifications when building \mathcal{A}_e , unless this is strictly necessary. Denote by s_f the accepting state of \mathcal{A}_e . The *start state* of \mathcal{A}_e is s_S , where S is the axiom of G_k . We proceed as follows:

1. There exists an edge in \mathcal{A}_e from s_S to $s_{1_i^q}$, labeled by a , where $[i,]_i \in N_l^{(2)}$ and

$[i \rightarrow a \in P_k$, if either $S[i]_i^q \in E_e$ or there exists a path in $\mathcal{G}.e^S$ from S to $]_i^q$ that contains no vertex labeled by $]_j^q$, $[j,]_j \in N_l^{(2)}$, or by $[_k^{-t}$, $[k,]_k \in N^{(1)}$. We fix $S \rightarrow a]_i^q \in P_r$.

2. There exists an edge in \mathcal{A}_e from s_S to $s_{[i]_i^{qt}}$, labeled by a , and an edge from $s_{[i]_i^{qt}}$ to $s_{[i]_i^{qt}}$ labeled by b , where $[i,]_i \in N^{(1)}$, $[i \rightarrow a$, and $]_i^t \rightarrow b \in P_k$, if either $S[i]_i^{qt} \in E_e$ or there exists a path in $\mathcal{G}.e^S$ from S to $[i]_i^{qt}$ that contains no vertex labeled by $]_j^q$, $[j,]_j \in N_l^{(2)}$, or labeled by $[_k^{-t}$, $[k,]_k \in N^{(1)}$. We fix $S \rightarrow a[i]_i^{qt}, [i]_i^{qt} \rightarrow b]_i^{qt} \in P_r$.

3. There exists an edge in \mathcal{A}_e from $s_{[i]_i^q}$ to $s_{[j]_j^q}$, labeled by a , where $[i,]_i, [j,]_j \in N_l^{(2)}$ and $]_j \rightarrow a \in P_k$, if either $]_i^q]_j^q \in E_e$ or there exists a path in $\mathcal{G}.e^S$ from $]_i^q$ to $]_j^q$ that contains no vertex labeled by $[k]_k^{qt}$ or by $]_l^q$, $[k,]_k \in N^{(1)}$, $[l,]_l \in N_l^{(2)}$. If $i = j$, i.e., $]_i^q]_i^q$ is a self-loop in $\mathcal{G}.e^S$, then $s_{[i]_i^q}s_{[i]_i^q}$ is a self-loop¹³ in \mathcal{A}_e . We fix $]_i^q \rightarrow a]_j^q \in P_r$.

4. There exists an edge in \mathcal{A}_e from $s_{[i]_i^q}$ to $s_{[j]_j^{qt}}$, labeled by a and an edge from $s_{[j]_j^{qt}}$ to $s_{[j]_j^{qt}}$ labeled by b , where $[i,]_i \in N_l^{(2)}$, $[j,]_j \in N^{(1)}$, $[j \rightarrow a$, and $]_j^t \rightarrow b \in P_k$, if either $]_i^q]_j^{qt} \in E_e$ or there exists a path in $\mathcal{G}.e^S$ from $]_i^q$ to $[j]_j^{qt}$ that contains no vertex labeled by $]_k^q$, $[k,]_k \in N_l^{(2)}$. We fix $]_i^q \rightarrow a[j]_j^{qt}, [j]_j^{qt} \rightarrow b]_j^{qt} \in P_r$.

5. There exists an edge in \mathcal{A}_e from $s_{[i]_i^q}$ to $s_{[j]_j^{q'}}$, labeled by a , where $[i,]_i, [j,]_j \in N_r^{(2)}$, and $]_j \rightarrow a \in P_k$, if $]_i^q]_j^{q'} \in E_e$. If $i = j$ and $q = q'$, then $s_{[i]_i^q}s_{[i]_i^{q'}}$ is a self-loop in \mathcal{A}_e (because $]_i^q]_i^{q'}$ is a self-loop in $\mathcal{G}.e^S$). We fix $]_i^q \rightarrow a]_j^{q'} \in P_r$. Note that, it is also possible to have $i \neq j$ and $q = q'$ or $q \neq q'$ (with $i = j$ or $i \neq j$, case in which $]_i^q]_j^{q'}$ is a glue edge in $\mathcal{G}.e^S$).

6. There exists an edge in \mathcal{A}_e from $s_{[i]_i^q}$ to $s_{[j]_j^{q'}}$, labeled by a , where $[i,]_i \in N_r^{(2)}$, $[j,]_j \in N_l^{(2)}$, and $]_j \rightarrow a \in P_k$, if there exists a path in $\mathcal{G}.e^S$ from $]_i^q$ to $]_j^{q'}$ that contains no vertex labeled by $]_k^-$, $[k,]_k \in N_l^{(2)} \cup N_r^{(2)}$, or labeled by $[_l^{-t}$, $([l,]_l) \in N^{(1)}$. We fix $]_i^q \rightarrow a]_j^{q'} \in P_r$. Note that, q may be equal to q' .

7. There exists an edge $s_{[i]_i^q}s_{[j]_j^{q't}}$ labeled by a , and an edge $s_{[j]_j^{q't}}s_{[j]_j^{q't}}$ labeled by b , where $[i,]_i \in N_r^{(2)}$, $[j,]_j \in N^{(1)}$, $[j \rightarrow a$, and $]_j^t \rightarrow b \in P_k$, if there exists a path in $\mathcal{G}.e^S$ from $]_i^q$ to $[j]_j^{q't}$ that contains no vertex labeled by $]_k^-$, $[k,]_k \in N_l^{(2)} \cup N_r^{(2)}$, or by $[_l^{-t}$, $[l,]_l \in N^{(1)}$. We fix $]_i^q \rightarrow a[j]_j^{q't}, [j]_j^{q't} \rightarrow b]_j^{q't} \in P_r$.

8. There exists an edge in \mathcal{A}_e from $s_{[i]_i^{qt}}$ to $s_{[j]_j^{q'}}$, labeled by a , where $[i,]_i \in N^{(1)}$, $[j,]_j \in N_r^{(2)}$, and $]_j \rightarrow a \in P_k$, if $[i]_i^{qt}]_j^{q'} \in E_e$. We fix $]_i^{qt} \rightarrow a]_j^{q'} \in P_r$. Note that, it is possible to have $q = q'$ or $q \neq q'$ (in the last case $]_i^{qt}]_j^{q'}$ is a glue edge in $\mathcal{G}.e^S$).

9. There exists an edge $s_{[i]_i^{qt}}s_{[j]_j^{q't}}$, labeled by a , and an edge $s_{[j]_j^{q't}}s_{[j]_j^{q't}}$ labeled by b , where $[i,]_i, [j,]_j \in N^{(1)}$, $[j \rightarrow a$, and $]_j^t \rightarrow b \in P_k$, if there exists a path in $\mathcal{G}.e^S$ from $[i]_i^{qt}$ to $[j]_j^{q't}$.

¹³This case deals also with the situation when $]_i^q, [i,]_i \in N_l^{(2)}$, occurs in a loop in $\mathcal{G}.e^S$ composed of only left brackets in $N_r^{(2)} \cup N^{(3)}$, excepting $]_i^q$. A loop composed of only left brackets in $N_r^{(2)} \cup N^{(3)}$ is ignored when building \mathcal{A}_e .

that contains no vertex labeled by $]_k^-, [k,]_k \in N_l^{(2)} \cup N_r^{(2)}$, or by $]_l^{-t}, [l,]_l \in N^{(1)}$. We fix $]_i^{qt} \rightarrow a[]_j^{q't}, []_j^{q't} \rightarrow b]_j^{q't} \in P_r$. Note that, $]_i^{qt}$ may be equal to $]_j^{q't}$, i.e., $i = j$ and $q = q'$, i.e., the case of a loop in $]_i^{qt}$.

10. - For any final vertex labeled by $]_i^q, [i,]_i \in N_r^{(2)}$, or by $]_i^{qt}, [i,]_i^t \in N^{(1)}$, in $\mathcal{G}.e^S$, we add in \mathcal{A}_e a new edge $s_{]_i^q} s_f$, or $s_{]_i^{qt}} s_f$, respectively. In both cases, this is labeled by λ . We set in P_r a rule of the form $]_i^q \rightarrow \lambda$ or $]_i^{qt} \rightarrow \lambda$, respectively.

The new grammar $G_r = (N_r, T, P_r, S)$, in which the set of rules P_r is built as above, and $N_r = \{]_i^- | [i,]_i \in N^{(2)} \} \cup \{]_i^-, [i,]_i^- | [i,]_i \in N^{(1)} \}$ is a regular grammar generating a regular superset approximation for $L(G_k)$. Recall that, some of the brackets in N_r may also be \hbar -marked (by distinct symbols). It is easy to observe that $L(G_r) = \varphi(R_m)$, where φ is the homomorphism in the proof of Theorem 2.9.

Note that since the regular language in the Chomsky-Schützenberger theorem is an approximation of the trace-language, R_m depends on the considered context-free grammar in Dyck normal form. Hence, the refinement of the regular approximation depicted in this section is considered with respect to the structure of the grammar G_k in Dyck normal form, where by the *structure* we mean the number of rules and nonterminals composing G_k . As for $L = L(G_k)$ there exist infinitely many grammars generating it, setting these grammars in Dyck normal form other trace languages can be drawn, and consequently other regular languages, of type R_m , can be built. The best approximation for L is the regular language with fewer words that are not in L .

Denote by \mathcal{G}_L the infinite set of grammars in Dyck normal form generating L , by \mathcal{R}_m the set of all regular languages obtained from the refined extended dependency graphs associated with grammars in \mathcal{G}_L , and by $\mathcal{A}_L = \{ \varphi(R_m) | R_m \in \mathcal{R}_m \}$ the set of all superset regular approximations of L . It is easy to observe that \mathcal{A}_L , with the inclusion relation on sets, is a partially ordered subset of context-free languages. \mathcal{A}_L has an *infimum* equal to the context-free language it approximates, but it does not have the *least element*. Indeed, as proved in [2], [14], [15], and [16], there is no algorithm to build for a certain context-free language L , the simplest context-free grammar that generates L . Hence, there is no possibility to identify the simplest context-free grammar in Dyck normal form that generates L . Therefore, there is no algorithm to build the minimal superset approximation for L . Where by the *simplest* grammar we refer to a grammar with a minimal number of nonterminals, rules, or loops (grammatical levels encountered during derivations). Consequently, \mathcal{A}_L does not have the least element.

It would be interesting to further study how the (refined) extended dependency graphs, associated with grammars in Dyck normal form generating a certain context-free language L , vary depending on the structure of these grammars¹⁴, and what makes the structure of the regular language R_m (hence the regular superset approximation) simpler. In other words, to find a hierarchy on \mathcal{A}_L , depending on the structure of the grammars in Dyck normal form that generate L . These may also provide an appropriate measure to compare languages in

¹⁴For instance, how does it look the extended dependency graph associated with a *nonself-embedding* grammar in Dyck normal form, and which is the corresponding regular superset approximation. Note that, a context-free -nonself-embedding grammar always generates a regular language (since the language is finite).

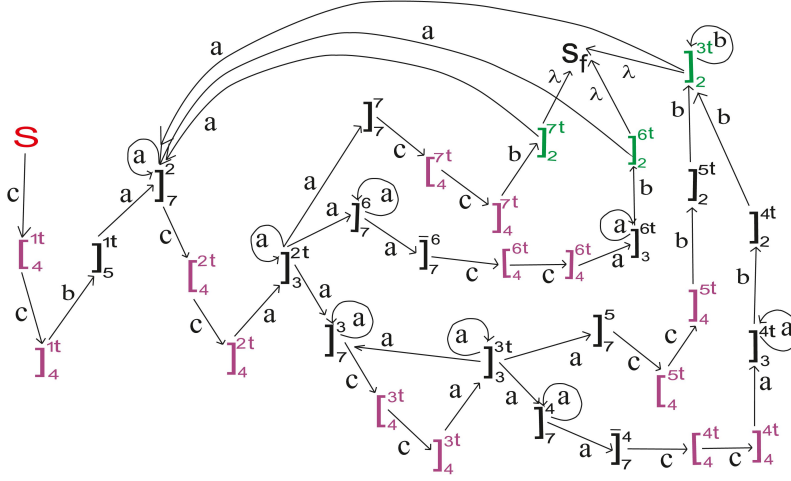


Figure 5: The transition diagram \mathcal{A}_e built from $\mathcal{G}.e^S$ in Example 4.2. Each bracket $[i (S, j_i)]$ in \mathcal{A}_e corresponds to the state $s_{[i} (s_S, s_{j_i})$ (see Example 5.1 b.). S is the initial vertex, vertices colored in green lead to the final state.

\mathcal{A}_L . On the other hand, for an ambiguous grammar G_k , there exist several paths (hence regular expressions) in the refined extended dependency graph, which “approximate” the same word in $L(G_k)$. Apparently, finding an unambiguous grammar for $L(G_k)$ may refine the language R_m . The main disadvantage is that, again in general, there is no algorithm to solve this problem. Moreover, even if it is possible to find an unambiguous grammar for $L(G_k)$, it is doubtful that the corresponding regular language R_m is finer than the others. In [14] it is also proved that the cost of the “simplicity” is the ambiguity. In other words, finding an unambiguous grammar for $L = L(G_k)$ may lead to the increase in size (e.g. number of nonterminals, rules, levels, etc.) of the respective grammar. Which again, may enlarge R_m with useless words. Therefore, a challenging matter that deserves further attention is whether the unambiguity is more powerful than the “simplicity” in determining a more refined regular superset approximation for a certain context-free language (with respect to the method proposed in this paper).

In [4] it is proved that optimal (minimal) superset approximations exist for several kind of context-free languages, but no specification is provided of how the existing minimal approximation can be built starting from the context-free language it approximates. It would be challenging to further investigate whether there exist subsets of context-free languages for which it would be possible to build a minimal superset approximation (by using the graphical method herein proposed).

Example 5.1. a. The regular grammar that generates the regular superset approximation of the linear language in Example 3.6 is $G_r = (\{S, [1,]_2^t, [3,]_4^t, [5,]_6^t, [7,]_7^t\}, \{a, b, c, d\}, S, P_r)$, where¹⁵ $P = \{S \rightarrow a[1,]_1 \rightarrow b[4,]_4 \rightarrow a[7,]_7 / a[7,]_7^t \rightarrow d[5,]_5^t \rightarrow c[3,]_3^t \rightarrow b[2,]_2^t \rightarrow c[3,]_3^t \rightarrow d[5,]_5^t, [2,]_2^t \rightarrow \lambda\}$. The language generated by G_r is $L(G_r) = \{(abb)^m aa(dcb)^n\}^p | n, m, p \geq$

¹⁵Note that, since there is only one dependency graph that yields only one plus-height regular expression there is no need of the labeling procedure described in Section 4.

$1\} = (abb)^+aa(d(cb)^+)^+= h(R)$. The transition diagram associated with the finite automaton that accepts $L(G_r)$ is sketched in Figure 1.c.

b. The regular grammar that generates the regular superset approximation of the context-free language in Examples 3.7 and 4.2 is $G_r = (\{S,]_2^{3t}, \dots,]_2^{7t},]_3^{2t},]_3^{3t},]_3^{4t},]_3^{6t},]_4^{1t}, \dots,]_4^{7t},]_5^{1t},]_7^2, \dots,]_7^7,]_7^6\}, \{a, b, c\}, S, P_r)$, where $P_r = \{S \rightarrow c[_4^{1t},]_4^{it} \rightarrow c[_4^{it},]_4^{jt} \rightarrow b[_5^{1t},]_4^{jt} \rightarrow a[_3^{jt},]_4^{mt} \rightarrow b[_2^{mt},]_5^{1t} \rightarrow a[_7^2,]_7^j \rightarrow a[_7^j,]_7^n \rightarrow c[_4^{nt},]_7^k \rightarrow a[_7^k,]_7^k \rightarrow c[_4^{kt},]_3^{2t} \rightarrow a[_7^3/a[_7^6/a[_7^7,]_3^{jt} \rightarrow a[_3^{jt},]_3^{3t} \rightarrow a[_7^3/a[_7^4/a[_7^5,]_3^{kt} \rightarrow b[_2^{kt},]_2^{lt} \rightarrow a[_7^2/\lambda,]_2^{ht} \rightarrow b[_2^{3t},]_2^{3t} \rightarrow b[_2^{3t}/a[_7^2/\lambda|h \in \{4, 5\}, i \in \{1, 2, 3, 4, 5, 6, 7\}, j \in \{2, 3, 4, 6\}, k \in \{4, 6\}, l \in \{6, 7\}, m \in \{5, 7\}, n \in \{2, 3, 5, 7\}\}$. The transition diagram associated with the finite automaton that accepts $L(G_r)$ is sketched in Figure 5.

6 Conclusions

In this paper we have introduced a normal form for context-free grammars, called *Dyck normal form*. Based on this normal form and on graphical approaches we gave an alternative proof of the *Chomsky-Schützenberger theorem*. From a *transition-like diagram* for a context-free grammar in Dyck normal form we built a *transition diagram* for a *finite automaton* and a *regular grammar* for a *regular superset approximation* of the original context-free language. A challenging problem for further investigations may be to further refine this superset approximation depending on the type of the grammar (e.g. nonself-embedding or unambiguous) or on the size of the grammar (e.g. number of nonterminals, rules, etc.) generating a certain context-free language.

The method used throughout this paper is graphically constructive, and it shows that *i.* derivational structures in context-free grammars can be better described through nested systems of parenthesis (Dyck languages), and *ii.* the Chomsky-Schützenberger theorem may render a good and efficient approximation for context-free languages. Furthermore, the method provides a graphical framework to handle derivations and descriptive structures in context-free grammars, which may be useful in further complexity investigations of context-free languages.

References

- [1] J. Berstel. Transductions and Context-Free Languages. Teubner, 1979.
- [2] A. Černý. Complexity and Minimality of Context-Free Grammars and Languages. *Proceedings of the 6th Symposium on Mathematical Foundations of Computer Science (MFCS 1977)*, Tatranska Lomnica, Czechoslovakia, Ed. Jozef Gruska, LNCS 53, 263–271, 1977.
- [3] R.S. Cohen. *Techniques for Establishing Star Height of Regular Sets*. Mathematical Systems Theory, vol. 5, 97–114, 1971.
- [4] B.J. Cordy, K. Salomaa. *On the existence of regular approximations*. Theoretical Computer Science, **387**(2), 125–135, 2007.

- [5] A. Cremers, S. Ginsburg. *Context-Free Grammar Forms*. Journal of Computer and System Sciences, **11**(1), 86–117, 1975.
- [6] O. Egecioglu. Strongly Regular Grammars and Regular Approximation of Context-Free Languages. *Proceedings of Developments in Language Theory, 13th International Conference (DLT 2009)*, Stuttgart, Germany. LNCS 5583, Springer 2009, 207–220.
- [7] L.C. Eggan. *Transition graphs and the star height of regular events*. Michigan Mathematical Journal, **10**(4), 385–397, 1963.
- [8] G. Eisman, B. Ravikumar. Approximate Recognition of Non-regular Languages by Finite Automata. *Proceedings of Twenty-Eighth Australasian Computer Science Conference (ACSC2005)*, Newcastle, Australia. CRPIT, 38. Estivill-Castro, V., Ed. ACS. 219–228, 2005.
- [9] A. Ehrenfeucht, H.J. Hoogeboom, G. Rozenberg. *Coordinated Pair Systems; Part I: Dyck Words and Classical Pumping*. Informatique Théorique et Applications, **20**(4), 405–424, 1986.
- [10] A. Ehrenfeucht, H.J. Hoogeboom, G. Rozenberg. *Coordinated Pair Systems; Part II: Sparse Structure of Dyck Words and Ogden’s Lemma*. Informatique Théorique et Applications, **20**(4), 425–439, 1986.
- [11] S. Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, 1966.
- [12] S.A. Greibach. *A New Normal Form for Context-Free Phrase Structure Grammars*. Journal of the Association for Computing Machinery, **12**(1), 1965, 42–52.
- [13] H. Gruber, M. Holzer. *Finite automata, digraph connectivity and regular expression size*. Technical report, TUM-I0725, Technische Universität München, December 2007.
- [14] J. Gruska. *Complexity and Unambiguity of Context-Free Grammars and Languages*. Information and Control, 18, 502–519, 1971.
- [15] J. Gruska. *On the Size of Context-free Grammars*, Kybernetika, **8**(3), 502–519, 1972.
- [16] J. Gruska. Descriptive complexity of context-free languages. *Proceedings of Symposium and Summer School on Mathematical Foundations of Computer Science (MFCS 1973)*, Strbské Pleso, High Tatras, Czechoslovakia, 71–83, 1973.
- [17] M.A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Longman, Inc., 1978.
- [18] K. Hashiguchi. *Algorithms for Determining Relative Star Height and Star Height*. Information and Computation, **78**, 124–169, 1988.
- [19] O.H. Ibarra, T. Jiang, B. Ravikumar. *Some Subclasses of Context-free Languages are in NC^1* . Information Processing Letters, **29**(3), 111–117, 1988.

- [20] P. Linz. An Introduction to Formal Languages and Automata. Jones and Bartlett Publishers, Inc., Sudbury, Massachusetts, 2001.
- [21] M. Mohri, M.J. Nederhof. Regular Approximation of Context-free Grammars Through Transformation. *Robustness in Language and Speech Technology*, vol. 9, 251-261. Kluwer Academic Publishers, 2000.
- [22] M.J. Nederhof. *Practical Experiments with Regular Approximation of Context-free Languages*. Computational Linguistics, **26**(1), 17–44, 2000.
- [23] J. Shallit, Y. Breitbart. *Automaticity I: Properties of a Measure of Descriptive Complexity*. Journal of Computer and System Sciences, **53**(1), 10–25, 1996.